THERMOMECHANICAL MODELING OF POROUS CERAMIC-METAL COMPOSITES ACCOUNTING FOR THE STOCHASTIC NATURE OF THEIR MICROSTRUCTURE

A Dissertation
Presented to
The Academic Faculty

by

Janine Johnson

In Partial Fulfillment
of the Requirements for the Degree
Ph.D. in the
School of Mechanical Engineering

Georgia Institute of Technology
May 2010

THERMOMECHANICAL MODELING OF POROUS CERAMIC-METAL COMPOSITES ACCOUNTING FOR THE STOCHASTIC NATURE OF THEIR MICROSTRUCTURE

Approved by:

Dr. Jianmin Qu, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Hamid Garmestani
School of Material Science
Georgia Institute of Technology

Dr. Arun Gokhale
School of Material Science
Georgia Institute of Technology

Dr. W. Steven Johnson
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Suresh Sitaraman
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Edgar Lara-Curzio
High Temperature Materials Laboratory
Oak Ridge National Laboratory

Date Approved: Nov. 5, 2009

To Dad and rousing renditions of “Here Comes the Sun”

ACKNOWLEDGEMENTS

I wish to thank Dr. Jianmin Qu, my PhD advisor, for his support and guidance through the years. I'm a better person for it. I would also like to thank my reading committee for all of their advice, data, and patience (and willingness to travel). Thank you Shenjia Zhang for the anode data. Next I cannot forget my editing friends. Thank you Sara for countless commas and Jeffrey Donnell for working very quickly. All of my labmates deserve thanks, but most especially Narasimhan for being a great person to talk OOP with. I've done just a tiny bit of computing, so a huge thanks to the support staff of the PACE and JADE computer clusters. Thank you Nitin, Lauren, Belen, Ashley, Christine, Charlotte, Temsiri and Caroline for listening to whatever I was worked up about at the time. Last, but not least, to several years of Georgia Tech friends and colleagues. I would name you all, but there just isn't enough space.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
CTE	Coefficient of Thermal Expansion
FE	Finite Element
LRO	Long Range Order
OOF	Object Oriented Finite
ORNL	Oak Ridge National Laboratory
pSOFC	Planar Solid Oxide Fuel Cells
PEN	Positive-Electrolyte-Negative
RVE	Representative Volume Element
SAM	Simulated Annealing Method
SERVE	Statistically Equivalent Representative Volume Element
SOFC	Solid Oxide Fuel Cells
SRO	Short Range Order

TPB	Triple Phase Boundary
VC-FEM	Voronoi Cell-Finite Element Method
YSZ	Ytrria Stabilized Zirconia

SUMMARY

Porous ceramic-metal composites, or cermets, such as nickel zirconia (Ni-YSZ), are widely used as the anode material in solid oxide fuel cells (SOFC). These materials need to enable electrochemical reactions and provide the mechanical support for the layered cell structure. Thus, for the anode supported planar cells, the thermomechanical behavior of the porous cermet directly affects the reliability of the cell. Porous cermets can be viewed as three-phase composites with a random heterogeneous microstructure. While random in nature, the effective properties and overall behavior of such composites can still be linked to specific stochastic functions that describe the microstructure. The main objective of this research was to develop the relationship between the thermomechanical behavior of porous cermets and their random microstructure. The research consists of three components. First, a stochastic reconstruction scheme was developed for the three-phase composite. From this multiple realizations with identical statistical descriptors were constructed for analysis. Secondly, a finite element model was implemented to obtain the effective properties of interest including thermal expansion coefficient, thermal conductivity, and elastic modulus. Lastly, nonlinear material behaviors were investigated, such as damage, plasticity, and creep behavior. It was shown that the computational model linked the statistical features of the microstructure to its overall properties and behavior. Such a predictive computational tool will enable the design of SOFCs with higher reliability and lower costs.

CHAPTER 1

INTRODUCTION

In a composite, a combination of different microstructures results in a hybrid material that may behave differently from any of the constituent materials. In metal-ceramic composites, called cermets, the strength of the ceramic plus the ductility of the metal may result in a stronger and less brittle material than a pure ceramic. The addition of a third phase, such as porosity, could increase a cermet's functionality by allowing fluid flow, while simultaneously reducing strength. This simple example demonstrates how a composite's bulk behavior becomes an increasingly complicated relationship of desired functions, volume fractions, phase distributions, and phase properties. Research that enhances the understanding of these relationships would benefit development of composites and the technologies that use them.

For instance, the study of porous cermets can be used in the development of planar solid oxide fuel cells (pSOFCs). All fuel cells have an anode and cathode layer for oxygen ion transfer across the electrolyte, and for fuel and air flow as shown in Figure 1.1. In pSOFCs the electrodes are solid components. In particular, the anode material (Ni-YSZ), made of a distribution of pores, nickel, and ytrria stabilized zirconia (YSZ), is a three-phase co-continuous cermet. Such three-phase materials are the focus of this work. The anode's proper function requires a continuous phase distribution, porosity for fuel flow, and the correct combination of ceramic and metal for structural support and electron transfer at high temperatures. Each phase in the anode must serve its own

functions without limiting the functionality of the other phases. For instance, as the volume percent of nickel increases, electrical and thermal conductivity will increase, but so will the thermal expansion of the microstructure. One way to study these increasingly complex interactions is through computer modeling of the microstructure.

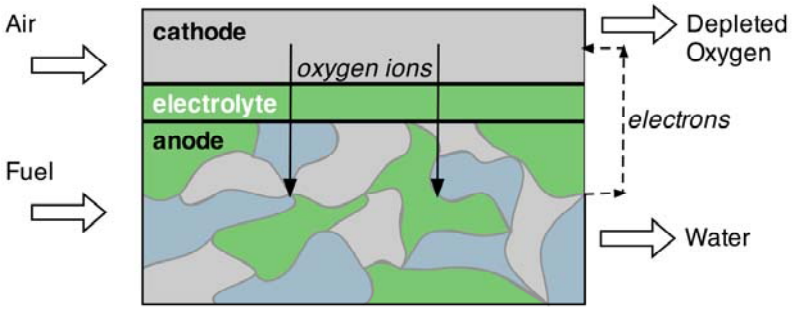


Figure 1.1. Schematic PEN layer of pSOFC.

Numerical models of co-continuous multi-phase composites must be robust enough to investigate multiple loading scenarios while accounting for the stochastic nature of the cermet. Each phase in the cermet is distributed such that each sample of the microstructure will be physically different, but all the samples can still be described with one set of stochastic functions. As a random heterogeneous microstructure, it is through stochastic descriptors, called probability functions, that the microstructure can be quantified. Plus, since it is not realistic to import multiple digital images of physical samples of Ni-YSZ, a method must be used to recreate realizations of Ni-YSZ from a unifying set of probability functions. A realization is defined in this context as a computer-generated image of the microstructure based on statistical descriptors of the material.

The simulations must also be three-dimensional (3D) since the co-continuous natures of the phases mean 2D models will not capture pore continuity and a disconnected material could result. Accurate determination of stress or temperature fields also requires a 3D analysis. Field behavior will follow the path of the microstructure depending on the loading conditions. In a 2D model those paths are limited. For instance, branching of stresses around pores will be limited to two paths, and a true picture of internal stresses will be missing.

The purpose of this research is to use stochastic reconstructions of the microstructure to model the material behavior of porous cermets, such as Ni-YSZ. The research consists of microstructure generation followed with a detailed study of material properties, damage plus plasticity, and creep. This is done through a combination of stochastic reconstructions and numerical simulations of thermomechanical properties with the purpose to enhance the overall understanding of porous cermets.

In the first step, a voxel-based stochastic reconstruction scheme was implemented to generate multiple realizations of the three-phase composite. Then, a finite element model was constructed for each of the microstructure realizations in order to conduct thermomechanical analyses that will allow us to obtain the effective properties of interest including thermal conductivity, elastic modulus, and thermal expansion. Stress-strain curves will be provided for plastic and creep behavior. Throughout this thesis, questions concerning the use of representative volume elements (RVEs) were addressed as the length scales increased from small scale deformations to large scale failure.

The rest of this chapter addresses the basic theory behind fuel cells, and current numerical methods to study them. The anode material is discussed in detail, and

operating conditions and the available experimental data are also discussed. The final sections will outline the research methodology and data results. Detailed theory and literature used in development of the research methodology are found in the chapters specific to each topic.

1.1. Fuel Cells

Society's increasing energy demands balanced with the need for cost effective and environmentally friendly fuels require that multiple avenues for power generation be investigated. To that end, fuel cells are a relatively established technology that uses electrochemical reactions to generate electric power with low environmental impact. While pSOFCs have significant promise for high power density applications, reliability and production difficulties limit their commercial viability [1, 2].

A planar SOFC consists of two porous layers (the anode and cathode), through which flow the fuel and oxidant, respectively (recall Figure 1.1). These metal-ceramic materials, or cermets, are bonded to a solid electrolyte layer to form a tri-layer structure called the PEN (positive-electrolyte-negative), across which ion diffusion generates a voltage. These electrochemical reactions take place within fuel cells at extremely high temperatures (>800ºC), which subject the cell components to harsh operating environments and severe thermomechanical stresses.

Recently, researchers have shifted focus to intermediate temperature pSOFCs, which operate at temperatures between 500-800ºC by changing the electrolyte to a thin film [1]. These intermediate temperature pSOFCs require less expensive interconnect materials and manifolds, and also reduce the thermal effects on the PEN. However, the use of thin film electrolytes shifts the functional support from the fully dense electrolyte

to the electrodes. In anode-supported cells, the anode material must now provide the necessary mechanical strength for the layered cell structure. Thus the thermomechanical behavior of the porous cermet, Ni-YSZ, directly affects the reliability of the cell.

Operating conditions of solid oxide fuel cells consist of three stages: start-up, steady-state, and shut-down. Initially, during start-up, the cell is heated from room temperature to the final operating temperature; many manufacturers send preheated air through the cathode during start up to reduce the chances of thermal shock. Steady-state operation occurs after the cell reaches its operation temperature. The operating temperature is influenced by several factors such as electrolyte thickness (thinner electrolytes enable lower operating temperatures) and the direction of fuel flow. During steady-state operation, stresses in the cell are influenced by the temperature gradients generated from the electrochemical reactions.

1.1.1. Computer Modeling

The study and optimization of SOFCs range from electrical output to structural degradation, but optimization of one parameter will impact, sometimes catastrophically, another aspect of the fuel cell. Numerical modeling then must take place at multiple levels to accurately capture these disparate factors. Models range from those that include the macroscopic behavior of the stack, the effective behaviors of laminated PEN layers and seals, and finally that of a specific component. Figure 1.2 is a simple illustration of these levels. Modeling at the component level would improve the usefulness of the macroscopic analyses by providing realistic approximations for different configurations and operating conditions.

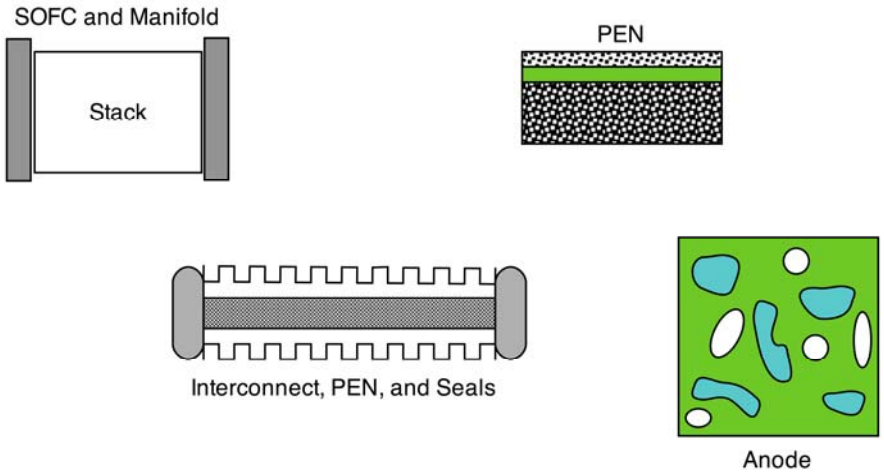


Figure 1.2. Multiple levels of SOFC modeling.

In the past, modeling has focused on determining the electrochemistry and thermal gradients occurring in the stack for various geometries and flow patterns [3-12]. Occasionally, these same models were used to find the stress gradients in the PEN layers [6]. Since a large fraction of cell failures are known to occur around seals and interconnects, several studies have recently focused on the PEN layer and a limited amount of the surrounding geometry [13-16]. Two of these studies have found the anode to be the main source of failure. Zhang et al. determined that fracture failure is most likely to occur in the anode layer of the PEN [13]. Nakajo et al. reached the same conclusion [14]. Studies of the anode material often focus on the linkage of the triple phase boundaries (TPBs), which is the point where all three phases conduct and is the active site for electrode reactions [17, 18]. Recently, Kim et al. has started using multi-level modeling to link the composite anode to the PEN level [18].

1.1.2. Anode (Ni-YSZ)

Ni-YSZ is one of the most commonly used anode materials in SOFCs [19], where each phase serves a specific purpose and the triple point boundaries are a site for electrochemical reactions. YSZ is added to help match the coefficient of thermal expansion (CTE) to the YSZ electrolyte and to prevent sintering of nickel. Material porosity varies from 20-40% for efficient mass transport. The nickel allows electron transfer. In anode-supported configurations, the anode is significantly thicker than the electrolyte layer and the two layers are often created by co-firing, which leaves residual stresses in the layers due to thermal mismatch of YSZ and Ni-YSZ.

This work examines the anode material manufactured by Oak Ridge National Laboratory (ORNL), which starts as a slurry of NiO and ZrO_2 stabilized with 8 mol% Y_2O_3 (Figure 1.3).

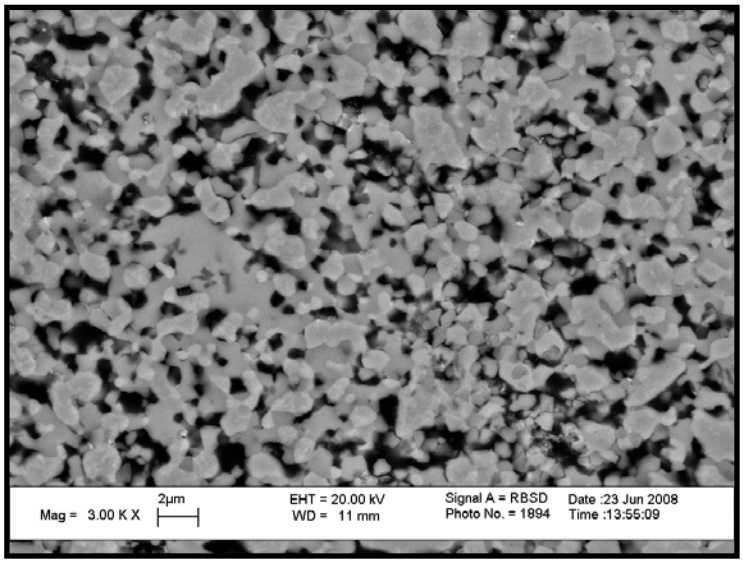


Figure 1.3. Image of 22 vol.% porous Ni-YSZ.

There is abundant experimental data for the electrical, thermophysical and mechanical properties of Ni-YSZ, but the data is tied closely to the specific composition and manufacturing of the given pSOFC. The electrical conductivity is often studied since it influences the performance of the fuel cell [20-23]. Another factor of interest is the reduction of oxygen in the anode precursor and the potential of re-oxidation in the anode [24, 25]. Thermophysical properties are significant as temperature-sensitive nickel interacts with temperature-resistant YSZ. The CTE and thermal conductivity both exhibit spikes in values at the Curie point of nickel [26, 27].

Elastic and fracture properties of both YSZ and Ni-YSZ have been studied in relation to temperature and porosity [25, 28-33]. Radovic et al. determined that the modulus of Ni-YSZ could be related to porosity using the composite sphere model [31]. Fracture toughness has been found to decrease with temperature and porosity, although the fracture toughness of Ni-YSZ was higher than YSZ [33]. Of significant interest is the long term temperature behavior of Ni-YSZ. Porosity and modulus were found to be independent of thermal aging although biaxial strength values decreased [34]. It was also found that the residual stresses in the electrolyte layer decreased with thermal aging, suggesting stress relaxation in the anode layer [34]. In one study by Gutierrez-Mora et al., creep in a bonded anode and electrolyte layer was diffusion-controlled at high temperatures [35]. Since the experimental data described above is specific to the manufacturing and composition of Ni-YSZ, a numerical study would provide insight into universal factors that dictate behavior.

1.2. Research methodology

The study of porous Ni-YSZ using reconstructions is a three-step process, where each step overlaps and influences the next. Initially a microstructure is created using a stochastic reconstruction, and then a FE analysis is implemented. This first round of FE was used to test model validity and to determine possible alterations to the microstructure. The objective is to link the probability functions to specific changes in the FE analysis. Detailed theory and literature for each step is provided in the relevant chapter.

1.2.1. Stochastic reconstruction

A composite can be reconstructed with three sets of information: the amount of each phase, the shape and size of particles, and how those particles interact. A 1D example, illustrated in Figure 1.1, starts with volume fraction, then length of particles, and finally how the particles are interrelated. In stochastic materials, amount, shape and interaction can be described through probabilities. The most basic descriptor is the volume fraction, φ_i, which is the ratio of the volume of phase i to the total volume of the material, and provides the amount of each phase. Volume fraction is also the probability that a given point will lie in the phase i. The information of each volume fraction could be used to recreate a simple realization of the microstructure. By adding more information to the probabilistic description, a progressively more detailed reconstruction is formed. By adding location, orientation, multiple sampling points, and other conditions, information about length scales of particles and their interactions is built into the probability function.

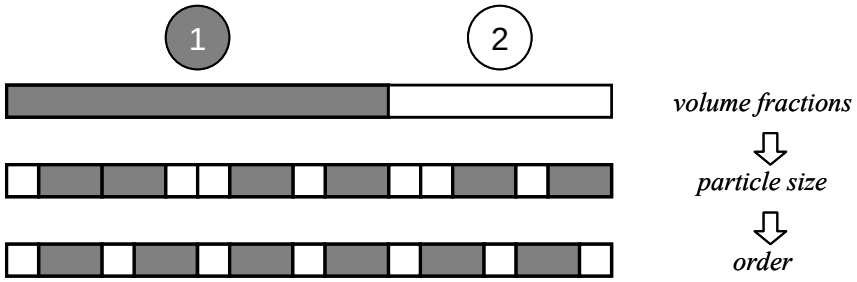


Figure 1.4. 1D visualization of composite construction.

The digital reconstruction method in this work uses probability functions of an ORNL sample to place voxels (3-D cubes) to create a realization [36]. The experimental results used to determine the probability functions of the ORNL sample showed an isotropic material, without any short range order (SRO) and a random long range order (LRO) [37]. SRO relates to the shape and size of particles in a composite and the lack of any definable order means the particles overlap. LRO tells how particles interact at a greater distance and for random materials is a constant. Nickel has a slightly larger particle size, or characteristic length, but other than the difference in volume fractions, 22% porosity and 43% YSZ, the pore phase and YSZ phase were identical. All phases percolate. Once the general behavior of the microstructure is understood, these probability functions can be used to create new microstructures for analysis.

1.2.2. Finite element models

The FE models can be grouped into three categories: structural, transport, and nonlinear behavior. Nonlinear behavior involves the incorporation of damage, plasticity and creep. Table 1.1 lists each area studied and the objective of the analysis.

Table 1.1. Numerical analyses and objectives.

Category	Field Behavior	Objective
Stochastic	percolation	• cluster size • relate volume fraction to percolation
FE Analysis		
Structural	elastic constants	• temperature dependence • porosity dependence • statistical variation of properties
Transport	thermal conductivity	• impact interfacial resistance
Damage and plasticity	stress-strain curves yield stress	• damage size and location in relation to microstructure • importance nickel length scale
Creep	strain-rate curves stress relaxation	• structural strength of Ni-YSZ versus strain rate • change in internal stresses • impact YSZ percolation

A robust reconstruction procedure allows creation of any number of realizations of the original ORNL sample and any amount of variations on the initial probability functions. Therefore each analysis will follow the same general principles. First the validity of the analysis will be assessed, followed by an extensive examination of a base model. The base model will either be the original ORNL microstructure or one with the same ratio of nickel to YSZ, but varying porosity. Once the base model is examined, the original probability functions are modified. New realizations are now examined to see how bulk behavior changes with alterations in the probability functions.

The validity of the FE models was examined by primarily looking at the discretization error and representative volume element (RVE) size. Discretization refers to the physical size of the voxel used for the realization, while representative volume element size is the total size of the realization. A discretization parameter and RVE size parameter are introduced to determine convergence of the FE models. To test this a

comparison to experimental data is used when available. An examination of field behavior with the microstructure might also be used to determine the correct voxel and volume size.

All finite element modeling is performed using the commercial software Abaqus 6.8.1. Post-processing of the FE models is performed using the scripting language Python. The models are constructed of eight node block elements with perfect bonding. Each voxel of the reconstruction is equivalent to one FE element, which is set to either the pore, YSZ, or nickel phase.

1.3. Summary

Porous cermets are of vital importance in the development of pSOFCs, and studies have found the anode material to be the material most likely to fail in anode-supported structures. Since the fuel cell undergoes high temperatures and stress gradients from thermal mismatch and construction, many different factors may play into this failure. It can be useful in pSOFC research to determine how changes in bulk behavior can be linked to changes within the composite. For example, when does nickel creep lead to a lack of structural support for the PEN, or which has more impact on the modulus: porosity or temperature?

The nature of the microstructure, three-phase and co-continuous, requires a three-dimensional reconstruction process. Rather than inputting multiple images of different anodes, a reconstruction process that can create multiple realizations for varying probability descriptors would provide insight into the microstructure. Voxel reconstruction is a computationally inexpensive and flexible method to do this, especially

considering it is also a 3D reconstruction. These realizations can then be used in multiple ways to study the porous cermet.

The following chapter will introduce the stochastic reconstruction in detail, as well as different probability functions that can be used for reconstruction. Chapter Three will introduce the cluster function, which can be used to determine percolation in the microstructure. The percolation becomes a parameter that can be directly correlated to various material properties. Chapter Four begins the FE analysis with an extensive examination of structural properties. Significant time is spent on determining the accuracy of the model, and then the impact of porosity and internal length scales are covered. In Chapter Five, the transport property, thermal conductivity, is examined, and a methodology to account for interfacial resistance in the FE model is covered. Chapter Six and Chapter Seven study nonlinear behavior in the microstructure. In Chapter Six, methods are introduced to study the occurrence of damage and plasticity in the structure. Chapter Seven focuses on stress relaxation due to creep in the nickel phase. General conclusions about porous cermets and the use of stochastic reconstructions are covered in the final chapter. Briefly covered will the possible impact for the study of pSOFCs.

CHAPTER 2

STOCHASTIC RECONSTRUCTION

In this work, the 3-phase composite, Ni-YSZ is numerically generated using a digitized stochastic reconstruction methodology. A three-dimensional realization is generated from cubic building blocks, termed voxels, distributed to match probabilistic functions describing each phase. The multiple realizations allow determination of the statistical variation of key material properties, such as modulus and the coefficient of thermal expansion (CTE).

Reconstruction of heterogeneous structures takes place down several avenues depending on the type of microstructure to be studied. Fiber and particulate composites can be reconstructed through tessellation procedures and then analyzed using self-consistent methods. Pyrz studied fiber composites using Dirichelet tessellations for simulated hard-core models and microscopy images, respectively, where stresses were calculated using reflection models [38]. Further research by Bochenek and Pyrz studied unidirectional fiber-reinforced composites and particulate composites using Voronoi tessellations, where each tessellation is treated as an element with a particle at its center. Ghosh et al. directly incorporated RVEs generated from Voronoi tessellations into a multi-scale finite element analysis [39-44]. The microstructural model is termed VC-FEM and models particulates in a matrix using Voronoi tessellations [39]. The same method was used for a multi-scale damage analysis in porous materials [41]. Additionally, three-dimensional models were created through stereological methods for particle reinforced metal matrix composites [40]. Next, the concept of statistically

equivalent representative volume elements (SERVE) was used with VC-FEM to study fiber and particulate composites that varied randomly within the microstructure [41-44]. Other methods, such as the "shaking" method, use extremely detailed images of particulate composites to exactly recreate embedded impurities or porosity [45-51]. The process of smoothing actual particulates in metals was incorporated with probability distributions to recreate the materials by "shaking" the particulates [49-51]. Other methods perform finite element analyses on the actual microstructure input via the open source software OOF. The OOF model can be used for different studies, such as Cannillo's and Carter's stochastic damage analysis on a polycrystalline microstructure [52]. Each of the methods has different advantages and disadvantages. Tessellation procedures are best used for fiber or particle composites, but not necessarily well suited for complex three-phase composites. While OOF exactly recreates a microstructure it does not have the capability to generate new microstructures with the same stochastic descriptors.

A modification of the simulated annealing method (SAM) is used here for its flexibility and efficiency in creating multiple realizations for numerical analysis. Rintoul and Torquato used the simulated annealing method to reconstruct a distribution of spheres using the radial distribution function [53]. Yeong and Torquato then applied the simulated annealing method to the recreation of digitized media [54]. Calculation time was minimized by limiting optimization of the two-point correlation function and the lineal path function in orthogonal directions and then updating the functions only along rows and columns with one to one pixel exchanges. However, Manwart and Hilfer noted that the time-saving device of using only orthogonal directions will introduce anisotropy

for microstructures displaying significant short range order [55]. The impact of short range order was reduced by adding additional sampling directions in the SAM procedure in work by Cule and Torquato [56]. Microstructural information from a 2D slice can also be used to construct a three-dimensional image as shown in Part II of Yeong's and Torquato's work on reconstructing random media [57]. Rozman and Utz used several techniques to improve the efficiency of the Monte Carlo reconstruction, including the Great Deluge Algorithm plus an additional criterion for "uphill" moves, limiting pixel changes to the interface, and calculating perturbations of the probability functions [58]. Johnson and Qu used the SAM method in addition to Rozman and Utz's modifications to recreate three phase 3D microstructures, the primary basis for the following work [36].

A 3D reconstruction process is required for study of Ni-YSZ due to the continuity of each of the phases, making it possible that a 2D reconstruction will not capture the structural strength of the microstructure. Since the Ni-YSZ is isotropic, 2D probability functions can be easily expanded to 3 phases and the SAM method provides an efficient methodology to recreate this 3D microstructure. The use of cubes allows many different realizations to be created, while keeping the model small enough to be studied numerically. Another advantage of SAM is the ease with which microstructural changes can be implemented in the reconstruction.

The chapter begins with an explanation of the theory behind random media and the introduction of the probability functions used for the reconstruction similar to that published by Johnson and Qu [36]. The reconstruction of the microstructure is then discussed in detail followed by an analysis of the accuracy of the reconstruction. Computational details such as expense and methodology will be described throughout.

2.1. Theory of random media

2.1.1. Indicator function

For any media of volume V_i the microstructure can be fully characterized by an indicator function

$$I^{(i)}(\tilde{x}) = \begin{cases} 1, & \text{if } \tilde{x} \in V_i \\ 0, & \text{otherwise} \end{cases}, \tag{2.1}$$

where i is the phase number, and $\tilde{x}$ is the vector location within the volume. The microstructure is assumed to be static and therefore is not a function of time.

The indicator function describes every possible point with a material, such that for any number, k, of phases the following equality holds,

$$\sum_{i=1}^{k} I^{(i)}(\tilde{x}) = 1. \tag{2.2}$$

2.1.2. Probability functions

The indicator function (2.2) allows any random media to be described by determining the probability of a desired event or occurrence. For example, an event of interest could be when multiple points lie within the same phase. Such an event is an example of the n-point probability function as illustrated in (2.3).

$$S_n^{(i)}(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n) = P\left(\mathrm{I}^{(i)}(\tilde{x}_1) = 1, \mathrm{I}^{(i)}(\tilde{x}_2) = 1, \ldots, \mathrm{I}^{(i)}(\tilde{x}_n) = 1\right), \tag{2.3}$$

where P indicates the probability that a given location lies within phase i. As the order n increases more microstructural details are captured.

If the probability distributions of a material are invariant with respect to location, the material is statistically homogenous and the material is ergodic. Plus, if the random media does not depend on the orientation of the vector positions, but only on the

magnitude of the distance between the points, it can be considered isotropic. In this case, the *n*-point functions now become functions of the distance between the points such that $r = \left| \tilde{x}_j - \tilde{x}_i \right|$.

Thus for homogenous media the *1*-point function will reduce to the volume fraction, φ_i, of the material,

$$S_1^{(i)} = P\left(\mathrm{I}^{(i)}(\tilde{x}_j)\right) = \varphi_i , \tag{2.4}$$

and the 2-point functions become functions of distance r,

$$S_2^{(i)}(r) = S_2^{(i)}(\tilde{x}_1, \tilde{x}_2) = P\left(\mathrm{I}^{(i)}\left(\tilde{x}_1\right) = 1, \mathrm{I}^{(i)}\left(\tilde{x}_2\right) = 1\right). \tag{2.5}$$

Bounds exist for the 2-point function in homogenous media as the radius reduces to zero or extends to infinity. These are

$$\lim_{r \to 0} S_2^{(i)}(r) = \varphi_i \text{ and} \tag{2.6}$$

$$\lim_{r \to \infty} S_2^{(i)}(r) = \left(\varphi_i\right)^2 . \tag{2.7}$$

In equation (2.6), $S_2^i(r)$ reduces to $S_1^i(r)$ as r decreases and the two points converge to each other. In equation (2.7), as the distances between the points increase, they are no longer spatially correlated and the 2-point approach becomes equivalent to calculating the $S_1^i(r)$ at two separate points as shown in Figure 2.1.

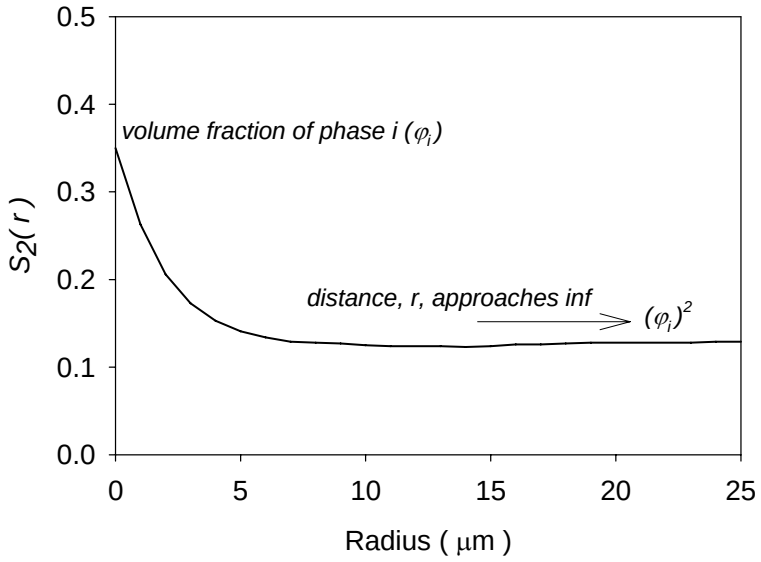


Figure 2.1. Example 2-point probability function.

2.1.3. Compatibility

The behavior of the 2-point probability function as r approaches zero provides a hint to the relationship between the probability functions for multiple phases. For a two-phase material, the description of one phase will guarantee its complement to the second phase, and Torquato and Stell showed that any *n*-point probability function can be written as a function of the other phases [22]. Thus as the number of phases increases the relationship must also be quantified between the phases, i.e. for three phases there are actually nine phases, namely phase 1, phase 2, phase 3, and permutations of any set of two phases. Therefore a complete set of probability functions fulfills the following equality for any distance, r, in (2.8).

$$\sum_{i=1}^{k}\sum_{j=1}^{k} S^{(ij)}(r) = 1 \tag{2.8}$$

However, if the reconstruction used all available probability functions to recreate a microstructure, it would be over-constrained, and only an independent set of three probability functions are needed. For a three-phase microstructure, this condition can be satisfied by using $S^{(1)}$, $S^{(2)}$, and $S^{(3)}$ (see proof in Appendix A based on [59]).

2.1.4. Alternate probability functions

The reconstruction does not have to be based on the use of *n*-point probability functions. Different characteristics can be captured by establishing different criteria. An example is the lineal path function. This function is equivalent to the two-point probability function except that now the path connecting the two points must lie in the same phase. Another example of function is the cluster function, which requires two points be connected within the same particle. These functions give some higher order information about connectivity of the phases, without the computational expense of *n*-point probability functions of $n > 2$. It should be kept in mind that the function definitions are arbitrary, and therefore cannot be treated in the same manner as *n*-point functions. These functions will be discussed further in Chapter 3.

2.2. Reconstruction methodology

The realizations were generated using the digitized simulated method introduced by Yeong and Torquato with modifications from Rozman and Utz [57, 58]. The algorithms were implemented in the C++ language and the GNU GCC compiler [60]. A complete set of the code can be found in Appendix B. The reconstruction procedure modifies the indicator function in equation (2.2) until the sample matches the desired probability functions. Then the indicator function is used to create a voxel representation

of the material for further numerical use. Each voxel represents a different phase. The algorithm is described in Figure 2.2.

1. Set sample size, voxel size, volume fractions, and reference probability functions ($f_{ref}^{(i)}$).
2. Used random number generation to seed indicator function.
3. Calculate probability functions ($f_{new}^{(i)}$) of current realizations in all three orthogonal directions.
4. Exchange random voxels located on interface. (A two-set paired voxel exchange is used to maintain volume fraction.)
5. Update probability functions using perturbation in each orthogonal direction.
6. Calculate energy (E)

$$E = \sum_i \sum_j \left[\left(f_{new}^{(i)}\left(r^j\right) - f_{ref}^{(i)}\left(r^j\right) \right)^2 \right]$$

7. Accept or reject change. Allow uphill movement through threshold value (∇)

$$P = \begin{cases} 1 & \text{if } E_{new} < \left(1+\nabla\right) E_{old} \\ 0 & \text{if } E_{new} > \left(1+\nabla\right) E_{old} \end{cases}$$

8. Repeat until an accepted criterion is reached.

Figure 2.2. Reconstruction algorithm.

The efficiency of the process was increased by using voxel selection at the interface, by sampling in orthogonal directions, using the great deluge algorithm, and using perturbations to calculate the correlation functions [36]. Recall that Manwart and Hilfer showed that sampling in orthogonal directions is acceptable for functions without short range order [55]. Boundaries are periodic.

Part of the flexibility of the DIB reconstruction is the ability to define an energy criterion based on desired probability functions. Any desired probability function can be used and these individual functions can be weighted as desired. This is because the energy function minimizes the least square difference between a given set of probability functions and the functions existing at the current time step of the reconstruction.

2.3. Ni-YSZ reconstruction

For this work, the three phases are described using the equation for Debye random media [61, 62]. For Debye media, the 2-point probability function does not exhibit any short-range order and is defined by one characteristic length, λ, and the volume fraction of the reconstructed phase, φ_i. The analytical expression for Debye media is shown in (2.8).

$$S_2^{(i)}(r) = \varphi_i(1-\varphi_i)e^{(-r/\lambda)} + (\varphi_i)^2 \tag{2.9}$$

These functions were found to match experimental data obtained from analysis of the anode material made at Oak Ridge National Laboratory (ORNL) [37]. Micrographs of the anode were obtained from a SEM (Joel 1530) with a pixel size of 0.05μm x 0.05μm. The characteristics of the ORNL sample are listed in Table 2.1.

Table 2.1. Parameters of ORNL Ni-YSZ.

i	**Phase**	φ_i	λ (μm)
1	Nickel	.35	.60
2	YSZ	.43	.40
3	Pores	.22	.40

Since nickel has the largest characteristic length, it is used to determine the length and size of the total reconstructed sample. Two parameters will be used for the reconstruction. The first parameter, N, will relate to the physical size of the sample and is a function of the representative volume length, L_{RVE}, and λ such that

$$N = \frac{L_{RVE}}{\lambda}. \tag{2.10}$$

The next parameter, R, relates the size of the realization to the voxel length, L_{voxel}.

$$R = \frac{L_{RVE}}{L_{voxel}}. \tag{2.11}$$

Accurate reconstruction of Ni-YSZ is based on correctly balancing these parameters, where N relates to physical size and R describes discretization as illustrated in Figure 2.3. As the voxel length decreases, with R increasing, more short range behavior is captured, and as the RVE length and N increase, more long range behavior is captured.

In future chapters, several sets of realizations are generated for varying sizes and compositions. However, in the rest of this chapter the ORNL sample, as described in Table 2.1, is used.

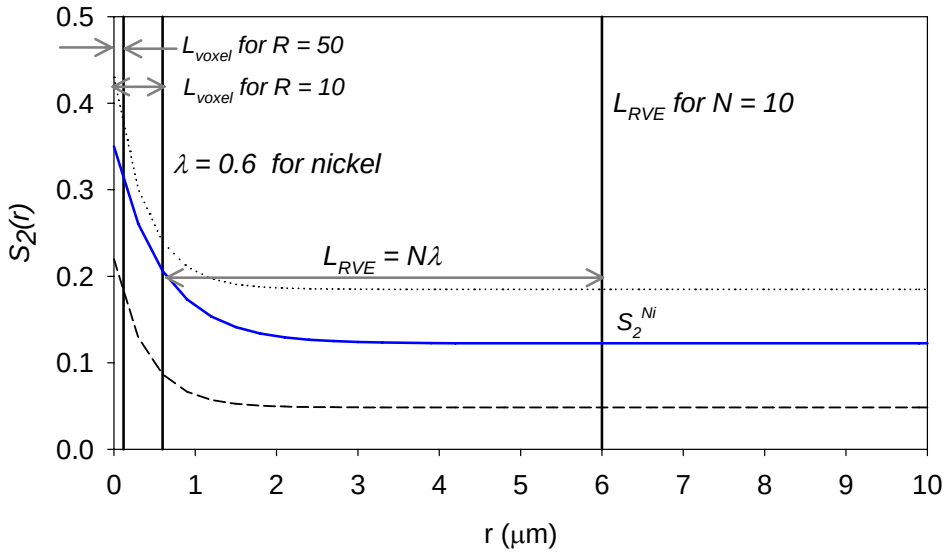


Figure 2.3. RVE length versus voxel length for to 2-point probability functions ORNL.

2.3.1. Two-point probability

Using equation (2.9) and the parameters listed in Table 2.1, the microstructure is reconstructed from a completely random distribution to a three-phase representation of Ni-YSZ. The reconstruction was done for multiple voxel and volume sizes, but only an R of 50 and N of 20 is illustrated in Figure 2.4.

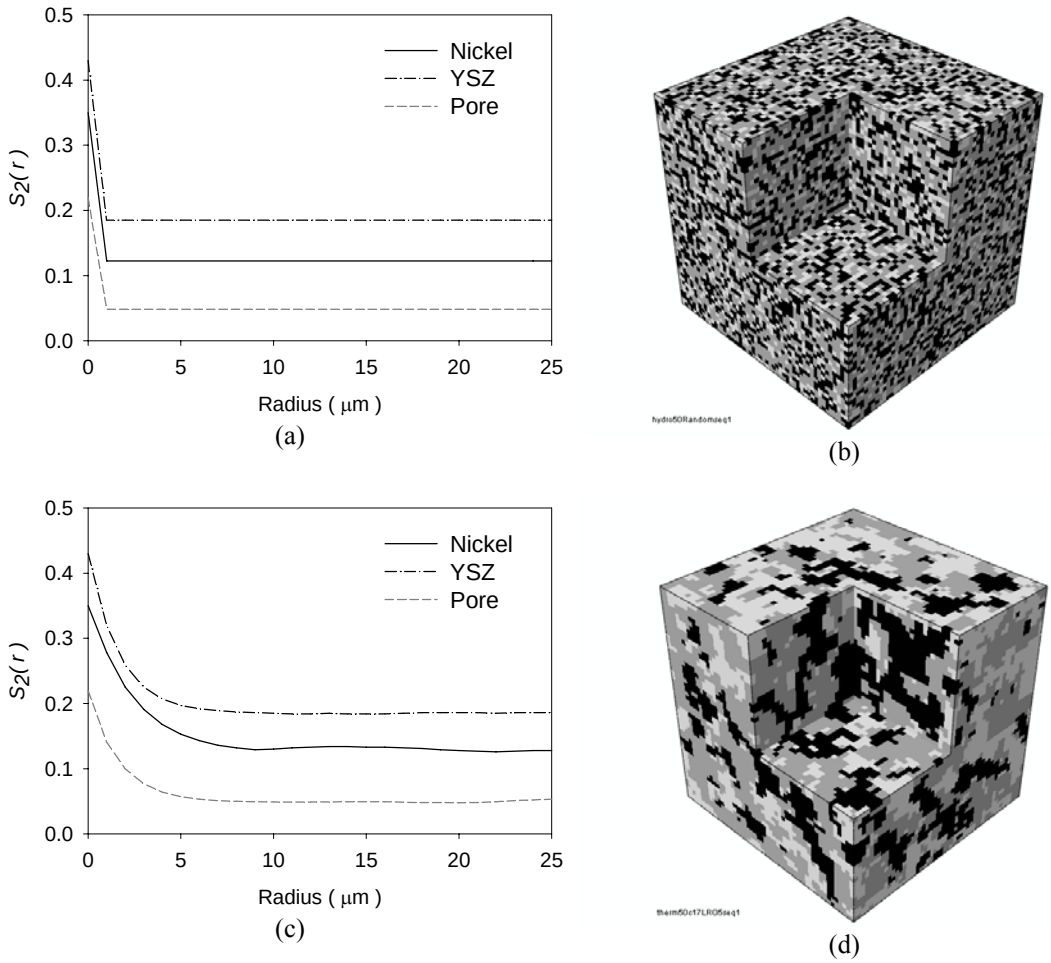


Figure 2.4. (a) Probability distribution for random microstructure (b) image of random distribution (c) probability distribution for final realization and (d) image of final realization (black – nickel, grey – YSZ, light grey – pores).

To confirm that the reconstruction has a realistic indicator function and phase relationships, all nine two-point probability functions were checked to determine whether equation (2.8) was satisfied. Figure 2.5 shows how the sum of functions will equal unity for every distance between the two points. Also, as the mixed phase probability functions extend toward infinity, they become equal to $\varphi_i \varphi_j$. This condition guarantees that each phase has random long range order (LRO). Both of these conditions ensure that the artificially generated indicator function accurately describes the composite.

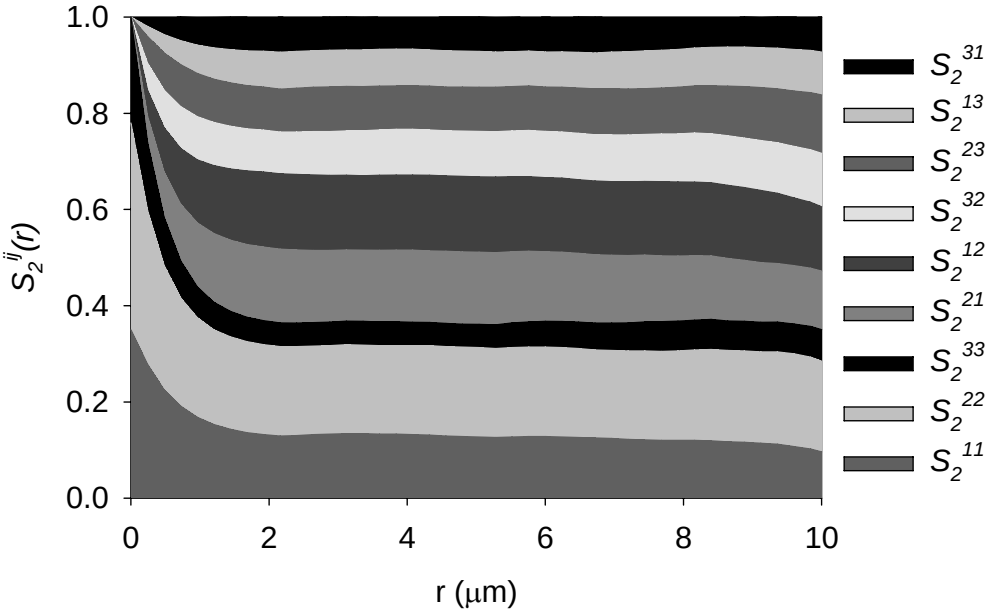


Figure 2.5. Sum of complete set of 2-point functions for Ni-YSZ.

2.3.2. Lineal path probability

Experimental data also provided lineal path functions for the microstructure [37]. The lineal path functions used in the reconstruction were best fit polynomial approximations from the experimental data. Examination of Figure 2.6 shows that the lineal path function for the nickel phase takes slightly longer to reach zero, corresponding with its larger characteristic length over the other two phases.

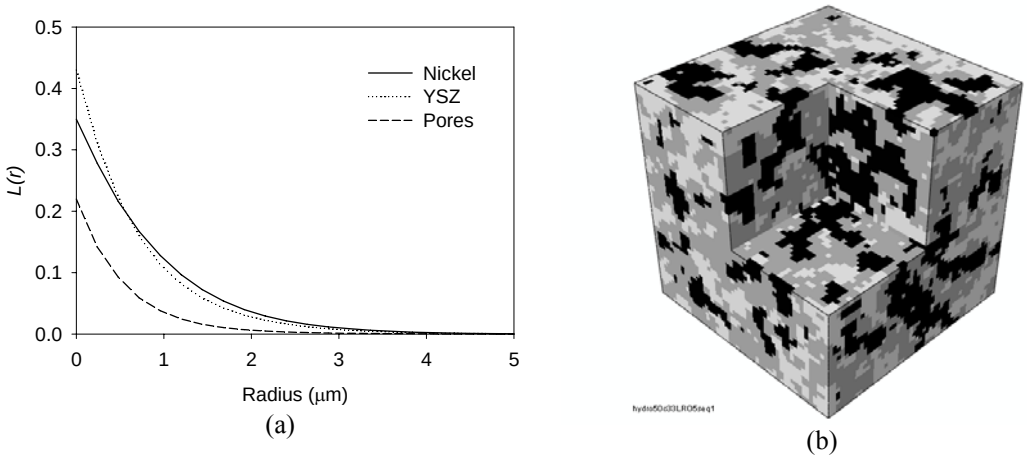


Figure 2.6. (a) Lineal path probability functions for Ni-YSZ and (b) image of lineal path based reconstruction.

2.4. Accuracy and computational expense

Two reconstructions have been generated, one based on the 2-point probability function and the other based on the lineal path function. Other than the material analysis in future chapters, two ways currently exist to differentiate between the reconstructions. These are the final energy of the realization (refer to Figure 2.2) and the computational expense of creating the realization.

During the reconstruction, the minimum acceptable energy, or least square difference, for the 2-point reconstruction was set to $1x10^{-7}$ for a given radius around the sampling point. For the lineal path analysis, the reconstruction would reach a global minimum well above this value, suggesting limitations in its ability to completely describe a microstructure. In Figure 2.7, the least square difference for the entire sample is totaled and compared between the two reconstructions. For the two-point reconstruction, the lineal path behavior is not captured. However, the lineal path reconstruction will accurately capture some minimum of the two-point probability

function, which is shown by the difference in scales between the figures. Primarily, the long range behavior of the microstructure is replicated using the lineal path function, while some of the short range behavior is lost.

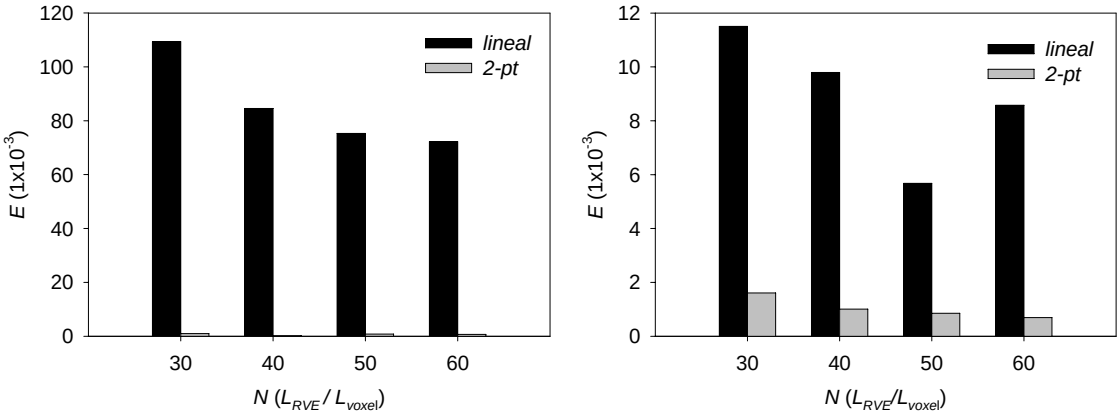


Figure 2.7. Difference between energies for final 2-point and lineal path probability functions for (a) the 2-point reconstruction and (b) the lineal path reconstruction.

The slight gain in microstructural detail with the lineal path function must also be balanced with the added computational expense. A primary advantage of the reconstruction methodology is the efficiency with which multiple realizations can be created and their microstructure modified. Figure 2.8 shows the exponential increase in reconstruction with volume size. The large increase is due to the nature of a 3-D reconstruction.

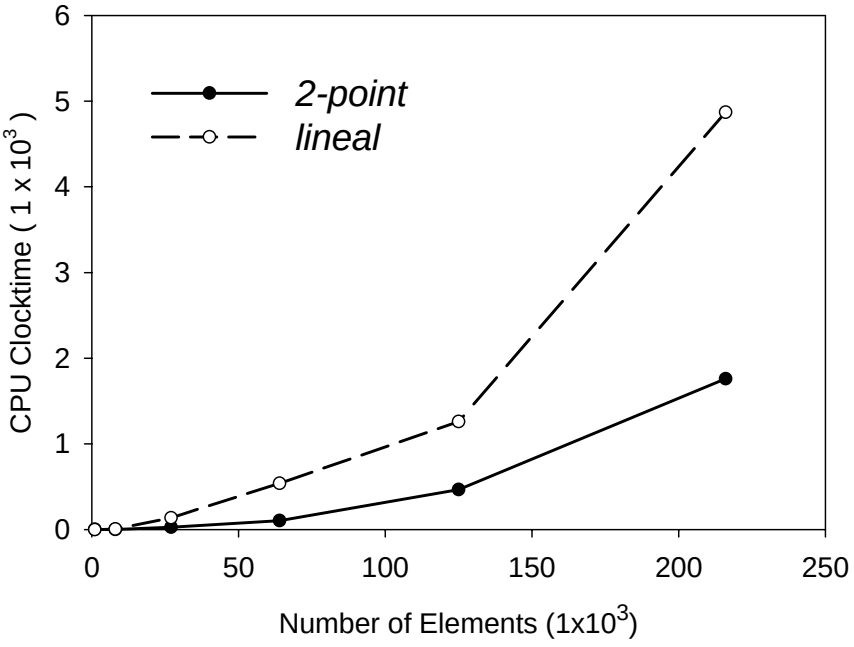


Figure 2.8. Computational expense of the reconstruction against number of elements.

2.5. Summary

The methodology for a voxel reconstruction of Ni-YSZ has been detailed and demonstrated for a realistic three-phase microstructure. The reconstructions were found to be compatible and a realistic representation of the microstructure. While the lineal path reconstruction captured more microstructural details than the 2-point reconstructions, the 2-point one was found to be much more efficient. The next chapter will further focus on connectivity by investigating percolation in the reconstructions. The final chapters will study the material behavior of the microstructure via finite element analysis.

CHAPTER 3

PERCOLATION

In complex composites, such as Ni-YSZ, the continuity of the phases will control fuel/air flow through pores, electron transfer in nickel, and the overall structural strength due to YSZ. Percolation of a system is a measure of connectivity, or clustering, but cannot be measured by lower order probability functions, such as the two-point function. Most significantly, the two-point probability function provides no insight into the percolation threshold. The percolation threshold, which Torquato defined as "the point where a cluster [connected group of elements] first spans the system", is a parameter which will tell us if flow can occur through a phase [63].

Torquato et al. used the two-point cluster function specifically to study percolating clusters in a composite system, and proved that the cluster function will become long ranged at the percolation threshold in a system [64]. Lee and Torquato numerically confirmed the long-range behavior at the percolation threshold for penetrable concentric-shell models [65]. Additionally, Lee and Torquato used series expansions of the mean cluster size to determine the percolation threshold, by calculating the point when the cluster size became infinite [66]. Percolation, as an infinite spanning cluster, is strongly influenced by both boundary conditions and window size. Sang Bub Lee reduced this impact using free boundary conditions for an optimized cell window, while Yi and Sastry treated percolation as a probabilistic property that approached unity as the window size increased [67, 68].

Percolation and its threshold for a given system is also scalable for given material properties, such as structural properties and transport properties like thermal conductivity. For instance, in a two-phase porous medium, there will be a point of percolation where the material will no longer be able to support loads, etc. In continuum systems, Feng et al. determined power relationship exponents for elasticity, conductivity, and permeability [69].

In this chapter, the cluster function is studied in relation to the 3-phase microstructure generated in Chapter 2. It is used to contrast the lineal path function reconstruction with the original Debye reconstruction. Finally, the impact of discretization is discussed and the percolation threshold is determined for the pore phase.

3.1. Theory

3.1.1. Probability functions

Two-point probability functions provide little information about how particles are connected within a microstructure. Higher order functions, such as the 3-point probability function, can provide this information, but are computationally expensive to use. Another approach is to add constraints on the 2-point function that will provide connectivity information such as the lineal path function defined in Chapter Two. Another function along those lines is the cluster function, which is defined heuristically in equation (3.1)

$$C_2^{(i)}(r) \equiv \frac{\text{probability that } x_1 \text{ and } x_2 \text{ can be connected}}{\text{by any line that lies entirely in phase } i} \tag{3.1}$$

While the development of the lineal path and cluster functions is not as rigorous as that seen for the n-point functions, bounds also exist. As r approaches zero eqn. (3.1) will also equal the volume fraction, representing a cluster of one particle.

3.1.2. Continuum percolation

In a multi-phase microstructure, the volume fraction, φ_i, of a given phase i is equivalent to the probability that a given point will lie in a cluster of any size. The percolation threshold, φ_{ic}, is the minimum volume fraction needed for a volume spanning cluster to first appear. This implies the following relationship,

$$\text{if } \varphi_{ic} > 0 \text{ then phase } i \text{ is continuous in the volume.} \qquad (3.2)$$

Determination of φ_{ic} is dependent on volume fraction, microstructural features, volume size, and so on. The final value is non-universal and is relevant only to a given specific microstructure.

Torquato et al. demonstrated that the cluster function will become long ranged at the percolation threshold such that [64],

$$\lim_{r\to\infty} C_2^{(i)}(r) = \varphi_{ic}, \qquad (3.3)$$

thus providing a way to determine when continuity occurs in any given phase for any set microstructure. For this work, the distance, r, where the cluster function begins to approach the percolation threshold is designated by $L_C^{(i)}$. It serves as a measure of clustering independent of percolation. The mean cluster size, Z_i, will be infinite in a continuum system that satisfies condition (3.2) and can be calculated from

$$Z_i = \frac{\eta_i}{\varphi_i^2} \int C^{(i)}(r) dr \,, \tag{3.4}$$

where η is the particle density.

3.2. Methodology

In the voxel reconstruction used here, each voxel is treated as a separate particle and is considered to be connected if any two particles share a full side. This provides the most conservative estimate of percolation possible. A cluster map is created that records each individual cluster for a phase and the location of each particle within it. For instance, in Figure 3.1, a total of eight clusters exist in the 2D representation. Each cluster is denoted with a white line. Then while sampling through the entire system, a two step criterion is used to determine the probability. First are any two points of the same phase (i.e. calculation of $S_2^{(i)}$), and second are the two points in the same cluster.

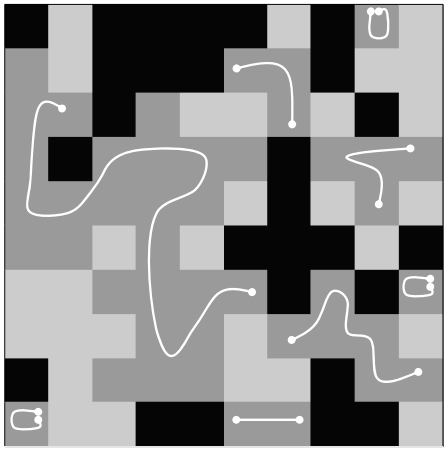


Figure 3.1. Illustration of voxel cluster in YSZ phase.

While orthogonal sampling is used for $S_2^{(i)}$, the cluster functions are not restricted to orthogonal planes. This is unlike the current methodology in calculation of the lineal path function, that samples only in three directions, and which would have binning complications if multiple directions or a purely random sampling was used. The curves are averaged across five realizations.

The percolation threshold is found when the change in percolation is less than 0.1% across the length. Boundary effects are neglected, and the periodicity of the microstructure ignored. To test the periodicity assumption, plot Figure 3.2 shows the cluster function for different measures of periodicity, where P states the number of voxels that are overlapped into the measurement. The x-axis distance between points is kept in voxels. While the curve changes, the overall change is small. Also, since periodicity is included, volume fraction is now capable of fluctuation. Therefore, to provide a more conservative estimate and to maintain volume fraction, periodicity is neglected.

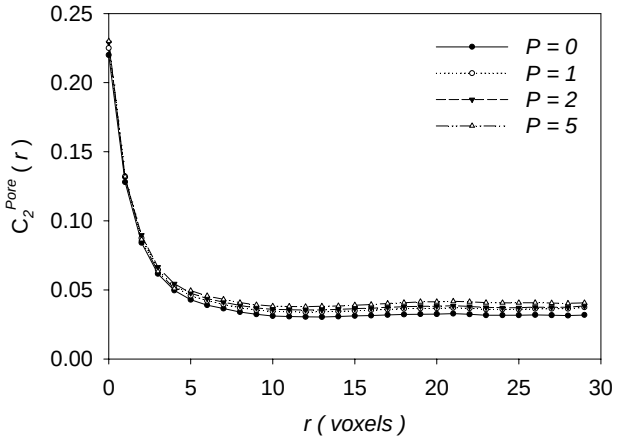


Figure 3.2. Impact of periodicity on clustering in pores.

3.3. Results

3.3.1. ORNL sample

In Figure 3.3, the cluster function, calculated without periodicity, is plotted for the ORNL sample reconstructed in Chapter 2. It can be seen that clustering follows behavior similar to that of the two-point functions. The lower long range order (LRO) is to be expected since all particles do not lie in one continuous cluster. The close match in short range order (SRO) happens, because for smaller distances, the points are more likely to connect. There is minimum difference in the nickel 2-point probability and cluster function, probably owing to the longer λ in this phase.

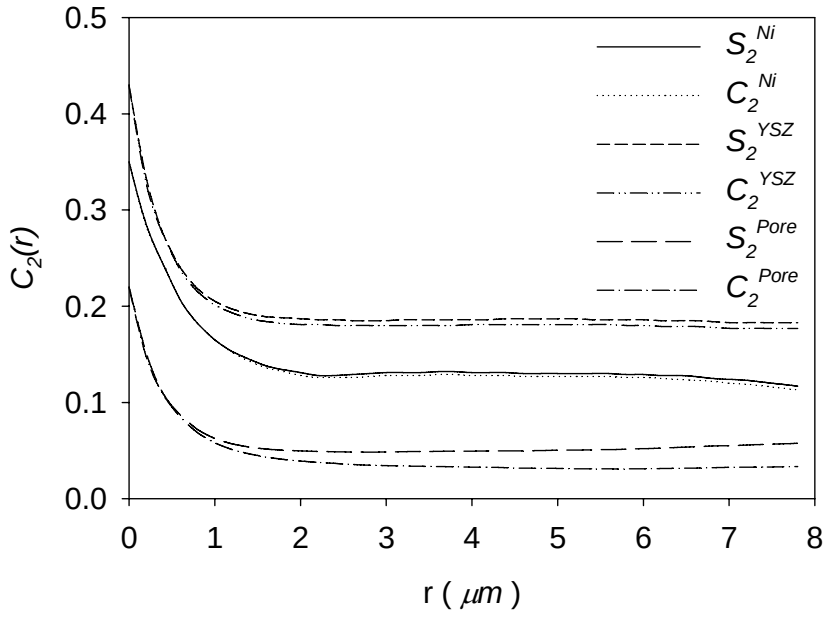


Figure 3.3. Comparison of $C_2^{(i)}(r)$ and $S_2^{(i)}(r)$ for ORNL sample.

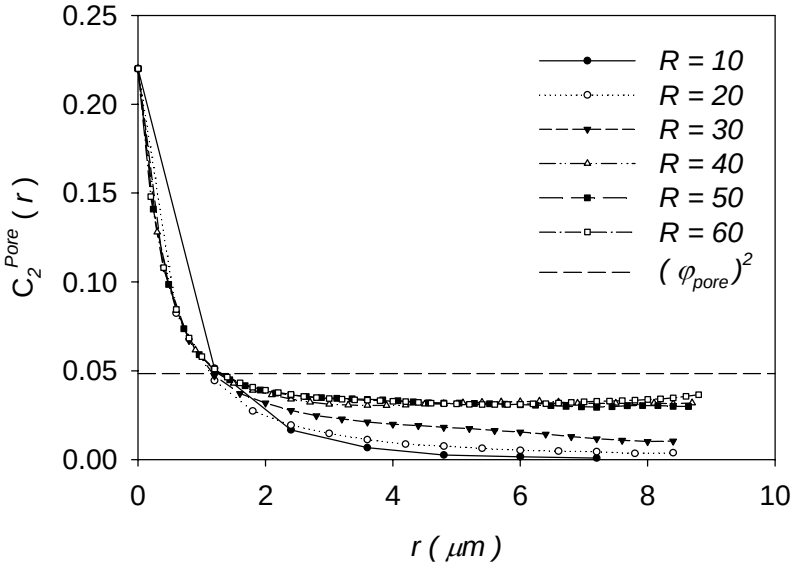
To test the convergence behavior of the cluster function, the pore phase for increasing R sizes, which measure discretization as a ratio of L_{RVE} to L_{voxel}, is plotted in

Figure 3.4. These functions are shown for both the two-point reconstruction and the lineal path reconstructions in Chapter Two. Also plotted for reference is the long range order (LRO) of S_2^{pore}. However, the lineal path function has a less consistent convergence behavior.

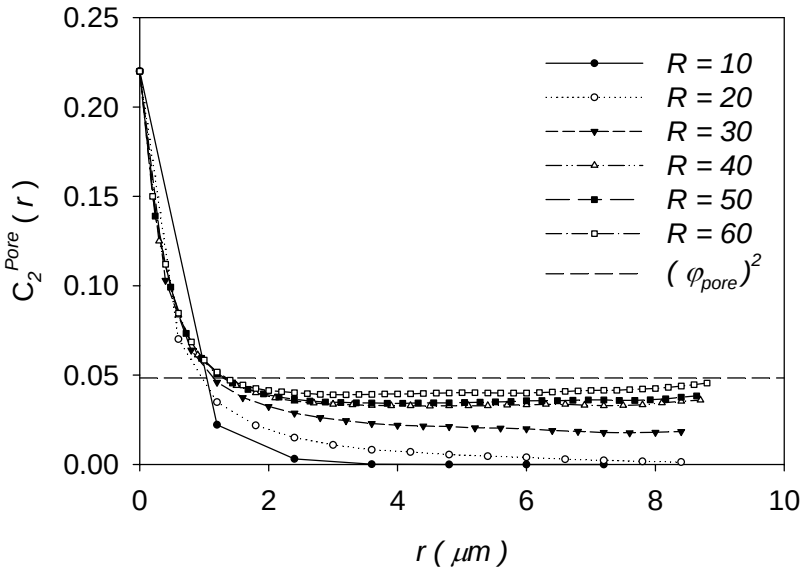
The lineal path reconstruction has different $L_C^{(i)}$ and Z_i sizes for nickel and YSZ as listed in Table 3.1 except for the pore phase. Since the average cluster size would be infinite for a continuous media, in this work it is calculated across three-fourths of the sample width, both to eliminate boundary effects and to capture more SRO behavior.

Table 3.1. Percolation values of ORNL sample for $R = 50$ and $N = 20$.

Model	**Phase *i***	$L_c^{(i)}$ **(μm)**	φ_{ic}	Z_i
2-Point	Ni	2.2	0.125	4.77
	YSZ	2.6	0.177	6.57
	Pore	2.9	0.032	1.57
Lineal	Ni	3.1	0.125	4.07
	YSZ	1.9	0.180	5.56
	Pore	2.9	0.034	1.57



(a)



(b)

Figure 3.4. Comparison of cluster function for (a) two-point and (b) lineal reconstructions based on ORNL sample.

3.3.2. Percolation threshold

Based on (3.2) it is reasonable to assume that when the cluster function reaches zero, flow will no longer occur within the phase for a given volume. As porosity decreases, the percolation threshold (eqn. (3.3)) and cluster size (eqn. (3.4)) also decreases. These values are plotted against varying porosities in Figure 3.5. For the realizations with 14% porosity, the phase no longer percolates and the average cluster size across the volume significantly drops. In Figure 3.6, the length at percolation, $L_C^{(i)}$, is plotted. The percolation threshold works well with a linear fit, and in the analysis, average cluster size matches a power curve. The fitting is done using the commercial software SigmaPlot and information on the equations can be found in Appendix C [70]. From the linear equation the percolation threshold equals zero at 16%. Both percolation threshold and average cluster size are closely correlated to porosity. The length at which percolation occurs, Figure 3.6, has higher variability, but is also independent of porosity, at least until the minimum threshold is approached.

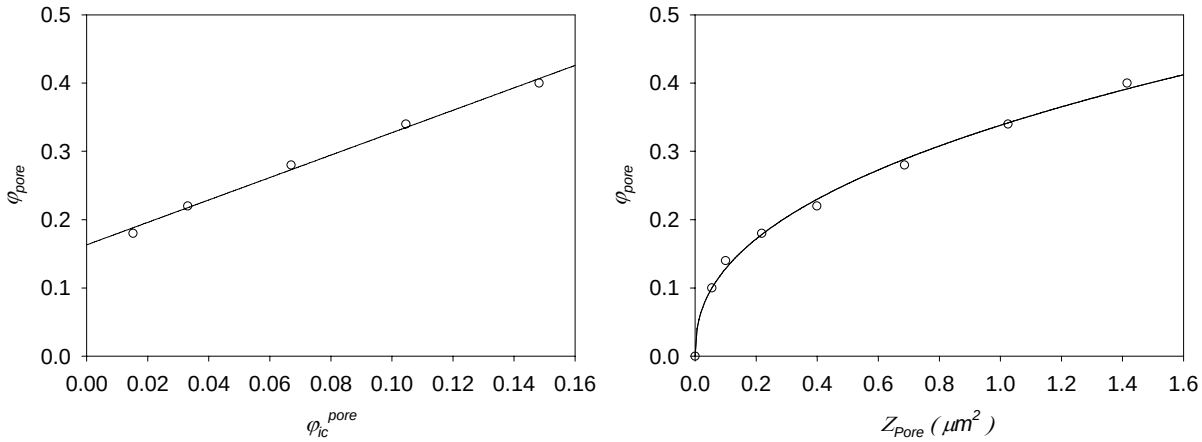


Figure 3.5. Percolation threshold (a) and average cluster size (b) as porosity decreases.

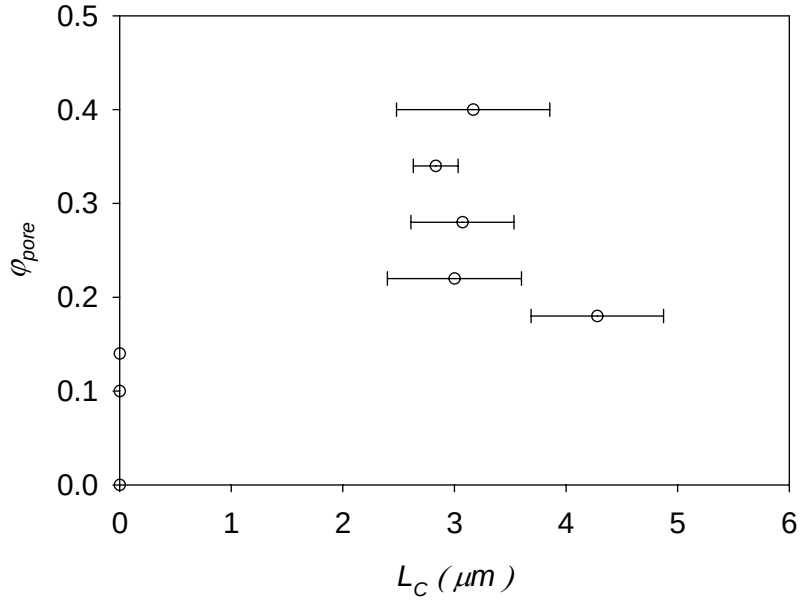


Figure 3.6. Cluster size across $12\mu m$ volume for different porosities.

3.4. Discussion

The cluster function introduced in (3.1) provided information about connectivity of the system through the percolation threshold and other parameters. It also differentiated between the continuity in different reconstructions. Figure 3.4 shows that an R greater than forty captures the long range behavior of the cluster function. At this point, the cluster function differentiates between the lineal path and two-point reconstructions in the different cluster sizes and lengths at percolation (refer to Table 3.1). The curves for porosity, however, are not significantly different. This probably relates to the porosity having a volume fraction of 22%, which means the smaller volume fraction dominates more than phase distribution. Figure 3.5 and Figure 3.6 prove that continuity will occur in the porous phase until 16% porosity and that L_C^{pore} is the only parameter independent of the volume fraction of porosity.

3.5. Summary

The percolation threshold and length have been introduced, as has a methodology for measuring them. These parameters, though specific to the reconstruction, can now be used as another measure of microstructure in future chapters. In the next chapter, percolation threshold and length will be shown to correlate with changes in modulus.

CHAPTER 4

EFFECTIVE STRUCTURAL PROPERTIES

Theories behind random heterogeneous materials use probabilities to characterize the distribution of phases within a microstructure, and as the phase distribution is stochastic, so will be the overall material response of the microstructure. Determination of effective properties becomes a function of the representative volume element (RVE), the length scales of heterogeneities, constituent properties, and the specific field behavior being studied. For any numerical analysis, computational error will also become a factor due to discretization of the microstructure. Therefore, before an accurate material analysis can take place, the necessary number of realizations and sufficient RVE size in relation to the microstructure must be determined. Once an RVE is established, changes in effective properties can be related to the volume fractions, microstructural details, and so on, allowing an efficient way to predict material behavior.

A major difficulty in the study of composites is that for every material behavior studied, the necessary RVE may change. A fiber composite can be easily studied with one 2D image of one fiber in a square, but as complexity grows, so will the RVE, Figure 4.1. Rules of thumb exist, such as that the RVE must be eight times larger than the largest characteristic length, yet examination of the literature shows no consistent formulation of RVE size [44, 46, 71-74].

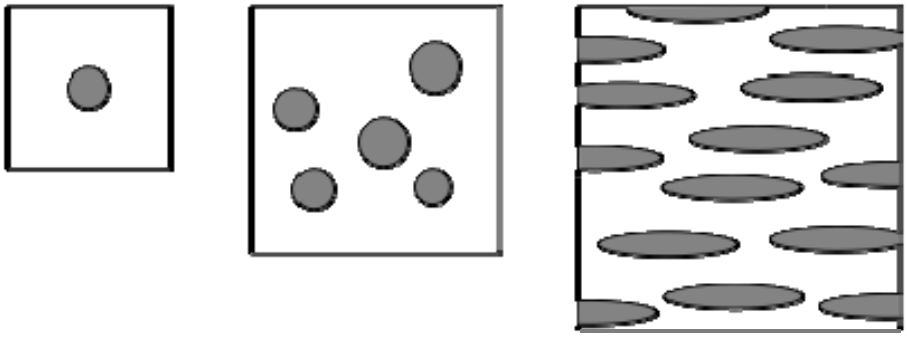


Figure 4.1. Examples of various RVEs.

Typically, material properties will be determined either through bounds or through analysis of a specific RVE. Bounds on material properties can be determined through variational methods such as Hashin-Shtrikman, which is based on volume fraction [75]. For greater accuracy, two-point and higher probability functions can also be used for bounds on material properties ranging from the effective elastic modulus, thermal conductivity, and even nonlinear plasticity [76-78].

In RVE analyses, FE methods are often used. The shape of the mesh depends on the reconstruction method, but could consist of regular shaped elements, or an irregular geometry dependant mesh. The microstructure is put into the FE model and then the constituent materials are controlled by the appropriate constitutive equations. Segurado et al. studied the effect of clustering on the total strength of metal matrix composites through a nonlinear FE analysis with a geometry dependant mesh [79]. Schmauder et al. used a plastic flow model to study residual stresses [80]. Shan and Gokhale created FE models from serial montaging and reconstructions to capture material behaviors at the desired length scales [47]. Examples of digitized, or regularly shaped FE elements, also exist for various mechanical analyses. Garboczi and Day used a linear elastic finite element model to determine bounds on the Poisson's ratio and other elastic properties for

any microstructure that can be digitized [71]. Terada et al. used digital images of fiber composites to look at microscopic stress variation and to determine the effective modulus and shear modulus [81]. Takano et al. used a voxel mesh to study the stress behavior in a porous microstructure [82]. Kumar et al. used the Yeong-Torquato reconstruction to perform an elastic-plastic analysis on multi-phase composites, which accurately modeled stress-strain curves and shear band localization [83]. Mishnaevsky and co-workers used voxel meshes to study damage growth in metals with particulates, graded composites, and porous microstructures [84-86].

In any FE analysis of composites, the first step is to determine RVE size. From a stochastic standpoint, the RVE is a volume sufficiently large to capture the statistical mean of the larger material [72, 73]. The RVE will vary in size and shape depending on the microstructure and application. Shan and Gokhale determined the smallest acceptable RVE sizes for fiber composites by using FE models containing different fiber sizes, volume fractions, configurations and different length scales [46]. Gitman et al. used FE to develop a stochastic criterion for representative volume elements [74]. A chi-square criterion was used to find lower size bounds by looking at different realizations and volume fractions. Knit et al. provided a quantitative definition of RVE size with the understanding that the material property measured will change the minimum size of the RVE [73]. Lastly, Swaminathan et al. investigated the size of statistically equivalent RVE's (SERVEs) using four different methods; 1) convergence modulus, 2) marked correlation functions, 3) distribution of significant microstructural features, and 4) two-point correlation functions [44].

Since it can be assumed the RVE will change for any material property, the work presented in this chapter focused on structural and expansion properties brought about by the stresses and strains occurring in the microstructure.

Initially, to study RVE size and discretization errors, each realization was converted into a FE model and appropriately modeled. Primarily, a graphical portrayal of the results is provided, but limited quantitative measures are used throughout to validate conclusions. This chapter focuses on the macro response of each realization, and detailed studies of field behavior within the microstructure were left for future chapters. The analysis was organized to meet the following objectives:

- To provide a "rule of thumb" in the determination of representative volume element size and appropriate voxel size.
- Determination of the mean and standard deviation of the effective property (assuming normality of data spread) with comparison to experimental data.
- Investigation into the appropriate number of samples and robustness of the analysis.

After determining the adequate RVE size, the microstructure was used to study increasing porosity. Then the impact of internal length scales on effective properties and clustering in the microstructure was studied. The impact of temperature also provided insight into how the bulk parameters, such as volume fraction and variation of constituent properties, affect the microstructure. This is an extension of work previously done by Johnson and Qu [36].

4.1. Theory

4.1.1. Representative volume elements

Representative volume elements (RVEs) must be of sufficient size to be statistically equivalent to the microstructure while still capturing the effective behavior independent of boundary effects or loading. In this work the RVE is represented by a cube shaped realization with sides of length, L_{RVE}, and volume, V. Boundary conditions are prescribed within the volume, or on the surface, δV, as illustrated in Figure 4.2.

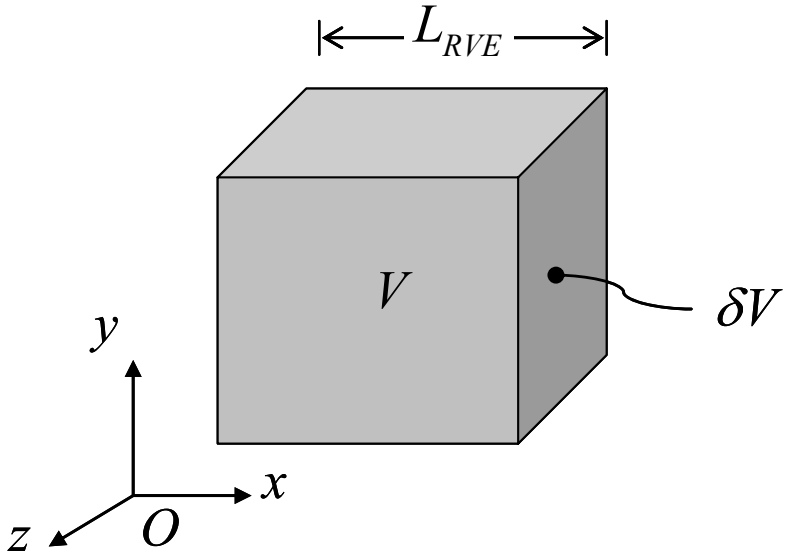


Figure 4.2. Schematic of representative volume element (RVE).

The volume average of any variable field value ($\mathbf{F}(x_i)$) occurring within the RVE is defined by

$$\langle \mathbf{F} \rangle \equiv \frac{1}{V} \int_V \mathbf{F}(x_i) dV , \tag{4.1}$$

where x_i is the position vector for a Cartesian coordinate system. The subscript i refers to indicial notation.

An additional notation will be used in this work that indicates a volume average within a specific phase in the microstructure, which is denoted by adding a phase suffix, $k = 1, 2,$ or 3.

$$\left\langle \mathbf{F}^k \right\rangle \equiv \frac{1}{V} \int_{V^k} \mathbf{F}(x_i) dV^k \tag{4.2}$$

The effective field behaviors for the RVE will be denoted by a superscript bar such as $\overline{\mathbf{F}}$.

Material properties such as Young's modulus, E, and Poisson's ratio, v, are isotropic composite properties unless a subscript is used to denote a specific phase.

4.1.2. Effective properties

The total effective strain, $\overline{\varepsilon}_{ij}^{tot}$, occurring within a composite in the elastic regime can be written as a sum of the elastic strain, $\overline{\varepsilon}_{ij}$, and thermal strains, $\overline{\varepsilon}_{ij}^{thm}$, such that

$$\left\langle \varepsilon_{ij}^{tot} \right\rangle \approx \overline{\varepsilon}_{ij}^{tot} = \overline{\varepsilon}_{ij} + \overline{\varepsilon}_{ij}^{thm} . \tag{4.3}$$

In equation (4.3), $\overline{\varepsilon}_{ij}^{tot}$ refers to the strains calculated from the composite modulus, E, and thermal expansion, α, and is equivalent to the volume average strain. The subscripts are indicial notation.

By assuming isotropy and Hooke's law, equation (4.3) can now be written as a function of the Cauchy stress (σ_{ij}), delta temperature (ΔT), the coefficient of thermal expansion (α_{mm}) and elastic constants (E, v) as shown in (4.4). δ_{ij} is the Kronecker delta.

$$\overline{\varepsilon}_{ij}^{tot} = \frac{(1+\nu)}{E}\langle\sigma_{ij}\rangle - \frac{\nu}{E}\langle\sigma_{mm}\rangle\delta_{ij} + \alpha_{mm}\langle\Delta T\rangle\delta_{ij}. \tag{4.4}$$

To solve for the constants in equation (4.4), three different load cases are required. Solving for CTE a temperature load is required such that,

$$\Delta T = C \text{ for } x_i \in V \tag{4.5}$$

and C represents any given constant.

The effective CTE is now

$$\alpha_{ii} \equiv \left\langle \varepsilon_{ii}^{tot} \right\rangle \Big/ \langle\Delta T\rangle. \tag{4.6}$$

Internal stresses are now developed within each phase, such that

$$\langle\sigma\rangle = \sum_{p=1}^{n}\left[\varphi_p \left\langle \sigma^p \right\rangle\right] = 0, \tag{4.7}$$

where φ_p is equal the volume fraction of each phase, p, for n phases.

The second case used a hydrostatic, or volumetric expansion, e, to determine the bulk modulus such that

$$e = \frac{dV}{V} = \langle\varepsilon_{ij}\rangle - \alpha_{ii}\langle\Delta T\rangle, \tag{4.8}$$

and where the displacements applied on δV must equal (4.9)-(4.11).

$$u_i = C_1 \text{ for } x_i = L_{RVE}, \tag{4.9}$$

$$u_i = 0 \text{ for } x_i = 0, \text{ and} \tag{4.10}$$

$$\Delta T = C_2 \text{ for } x_i \in V. \tag{4.11}$$

Equation (4.4) can now be solved for the bulk modulus,

$$K = \frac{E}{3(1-2v)} = \frac{\langle\sigma_{ii}\rangle}{3e}. \tag{4.12}$$

The assumption of isotropy is verified by ensuring that

$$\langle \sigma_{11} \rangle \approx \langle \sigma_{22} \rangle \approx \langle \sigma_{33} \rangle \text{ and} \tag{4.13}$$

$$\langle \sigma_{12} \rangle \approx \langle \sigma_{23} \rangle \approx \langle \sigma_{13} \rangle \approx 0 . \tag{4.14}$$

Stochastic variation in microstructure requires the approximation sign. Knit et al. showed that the coefficients of the full elasticity matrix converged to zero as RVE size increased [73].

Lastly, the modulus was solved by application of a fixed displacement in any given orthogonal direction as listed in equations (4.15) through (4.17).

$$u_i = C_1 \text{ for } x_i = L_{RVE} \text{ and} \tag{4.15}$$

$$u_i = 0 \text{ for } x_i = 0 \text{ for } i = 1, 2, \text{ or } 3 . \tag{4.16}$$

$$\Delta T = C_2 \text{ for } x_i \in V . \tag{4.17}$$

Now the temperature-dependant Young's modulus can be found using the following relationship.

$$E_{ii} = \frac{\langle \sigma_{ii} \rangle}{\langle \varepsilon_{ii} \rangle} = \frac{\varphi_p \langle \sigma_{ii}^p \rangle}{\langle \varepsilon_{ii}^{tot} \rangle - \bar{\alpha}_{ii} \langle \Delta T \rangle} . \tag{4.18}$$

In equation (4.18), as the thermal strain approaches the total strain for small loads, a singularity will occur, meaning that the effective modulus should be determined well away from small applied strains.

The combination of hydrostatic expansion and deformation in only one orthogonal direction can used to solve for shear modulus and Poisson's ratio through the following relationships,

$$G = \frac{3KE}{9K - E} \text{ and } \nu = \frac{3K - E}{6K} . \tag{4.19}$$

4.2. ORNL sample analysis

Since nickel has the largest characteristic length, refer to the ORNL sample in Table 2.1, as it is used to determine the length and size of the total reconstructed sample. The N and R parameters (refer to eqn. (2.10) and (2.11)) are used in the creation of multiple realizations. As previously defined, the parameter R is the ratio of L_{RVE} to L_{voxel} and provides information on the discretization of the microstructure, while N relates RVE size to the largest λ of the microstructure. The parameters N and R provide a dimensionless method of balancing the requirements of volume size and discretization of a voxel reconstruction.

Three additional models were also constructed for comparison purposes with the ORNL model: a random distribution, a periodic distribution, and finally a lineal path based reconstruction. The lineal path reconstruction used six order polynomial approximations of the lineal path functions determined from the ORNL sample and is plotted in Figure 2.6 with the same RVE size and voxel size as the realization in Figure 2.4(c-d). The random distribution in Figure 2.4(a-b) has the same volume fraction of the ORNL sample, but each voxel was placed using a random number generator. The periodic microstructure was arbitrarily set so the volume fractions are maintained and the RVE can be treated as a unit cell in recreating a larger structure, shown in Figure 4.3.

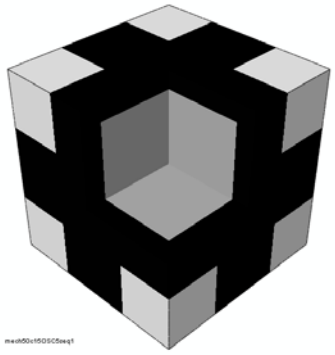


Figure 4.3. Unit cell RVE with volume fraction of ORNL sample.

4.2.1. Finite element model

The commercial software Abaqus was used to perform the finite element analysis. For each digital image reconstructed, the voxels were transferred to 8 node brick elements and input into a FE model. Perfect bonding was assumed between the elements, and each element was assigned the material properties corresponding to the digital reconstruction.

4.2.2. Constituent material properties

At room temperature the structural properties between nickel and YSZ are very similar; however, the temperature related properties of nickel are both significantly larger and more variable than YSZ. Table 4.1 lists the relevant material properties at room temperature and at 1000ºC (complete Abaqus property files are included in Appendix D). The pore phase had a modulus of 1 MPa and a CTE of 0 /ºC. Figure 4.8 also includes plots of CTE for YSZ and nickel. Experimental studies of the electrolyte material, non-porous nickel 8mol% ytrria (YSZ) material, were used as the YSZ properties in the Ni-YSZ composite [27, 33, 87].

Nickel is treated as a general polycrystalline material with primarily linearly dependant temperature properties. Grain size dependence is neglected. The behavior of nickel around the Curie point is complicated by the sudden jump in thermal expansion from the paramagnetic transition; therefore, a temperature dependent equation of CTE was used from the work of Faisst [88]. This expression was applied to published linear values to provide the CTE from 0ºC-1000ºC [89]. The change in modulus due to temperature for nickel is a linear relationship discussed by Kocks and Chen and commonly reported modulus values [90, 91].

Table 4.1. Comparison of constituent material properties.

Property	**Temp. (ºC)**	**YSZ**	**Ni**	**Ratio Ni to YSZ**
E (GPa)	25	216	207	1.0
	1000	216	121	0.6
ν	25	0.32	0.31	1.0
	1000	0.32	0.31	1.0
CTE (10^{-6}/ºC)	25	8.5	12.5	1.5
	1000	10.5	19.8	1.9

4.2.3. Convergence analysis

4.2.3.1. Discretization error and RVE size

Before determining the effective behavior of the modulus for varying microstructures, the accuracy and acceptable RVE size of the ORNL model are

determined. This is done by examining the effective material properties for varying discretization error and RVE sizes for the ORNL sample.

In Figure 4.4, box plots are provided for the modulus at two temperatures, as well as the CTE of the ORNL sample. The plots in the left column keep a constant volume, while varying R. The plots on the right column vary N while maintaining a constant element size. The line within each box plot is the median value of five samples with properties measured in the three orthogonal directions for a total of fifteen data points. The box itself represents the middle 50% of the data and the lines extend to the outliers.

In Table 4.3 and Table 4.4 the t-test and the chi-square criterion are used to compare the mean, χ, and standard deviations, S, between the samples for modulus data. This was done to validate the use of box plots to determine acceptable element and volume sizes. In Figure 4.1, Table 4.2 and Table 4.3, one particular volume is highlighted in grey. Even though each material property has different convergence behaviors, this volume was found to be acceptable for each analysis.

The convergence behavior of the lineal, random, and periodic reconstructions are shown in Figure 4.5. The mean values are plotted against a ratio of voxel size to characteristic length, which clearly illustrates the convergence behavior as voxel size decreases. Once a sufficient voxel and RVE length are reached, it can be seen that the results are independent of RVE size.

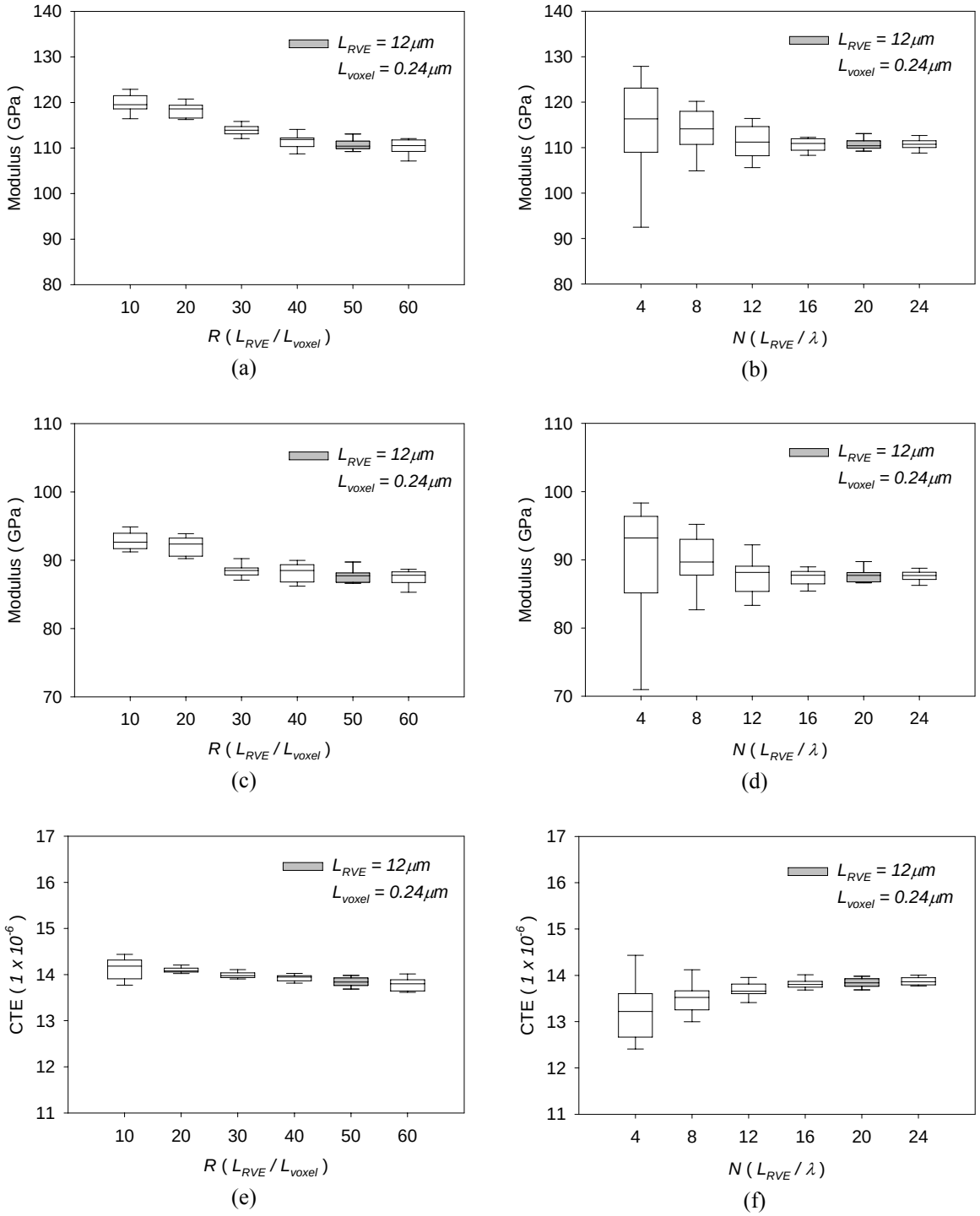


Figure 4.4. Box plots of discretization error and RVE size for Young's modulus at room temperature (a-b), Young's modulus at 1020ºC (c-d), and CTE at 1020ºC (e-f).

Table 4.2. Discretization of modulus at 25ºC.

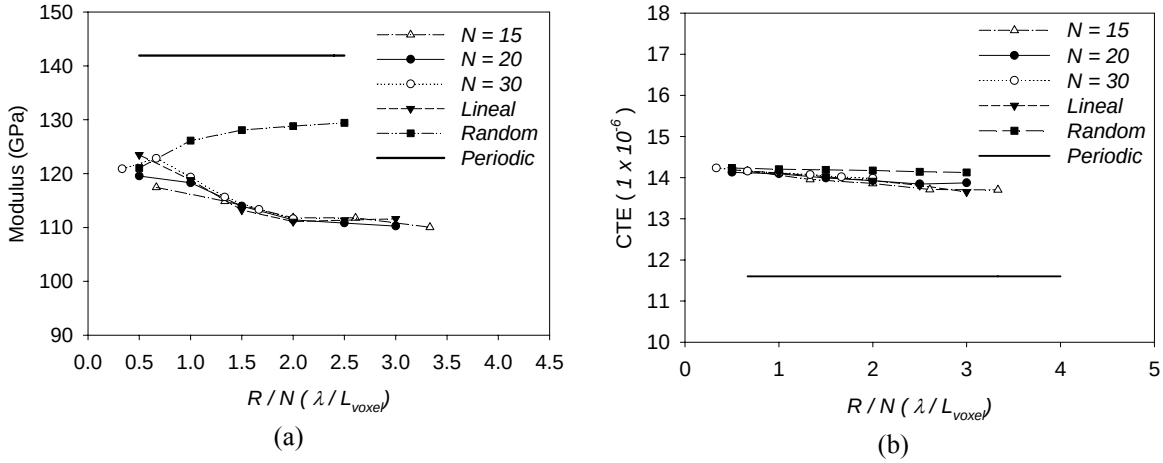
Model #	Num	R	N	L_{RVE} (μm)	L_{voxel} (μm)	χ (GPa)	S	t-test*	chi-square test**
ORNL	15	10	20	12	1.20	119.56	2.12	fail	fail
ORNL	15	20	20	12	0.60	118.29	1.63	fail	pass
ORNL	15	30	20	12	0.40	113.91	1.26	fail	pass
ORNL	15	40	20	12	0.30	111.42	1.71	pass	pass
ORNL	15	50	20	12	0.24	110.82	1.29	–	–
ORNL	15	60	20	12	0.20	110.28	1.78	pass	fail

*NULL Hypothesis – Mean of sample is equivalent to mean of Model #5 with failure to reject hypothesis at 5% significance level.
**NULL Hypothesis – Variance is equivalent to variance of Model #5 with failure to reject hypothesis at 5% significance level.

Table 4.3. RVE size of modulus at 25ºC.

Model #	Num	R	N	L_{RVE} (μm)	L_{voxel} (μm)	χ (GPa)	S	t-test*	chi-square test**
ORNL	15	50	4	2.4	0.24	114.71	13.38	pass	fail
ORNL	15	50	8	4.8	0.24	113.58	5.07	pass	fail
ORNL	15	50	12	7.2	0.24	111.09	3.70	pass	fail
ORNL	15	50	16	9.6	0.24	110.64	1.72	pass	pass
ORNL	15	50	20	12	0.24	110.82	1.29	–	–
ORNL	15	50	24	14.4	0.24	110.82	1.22	pass	pass

*NULL Hypothesis – Mean of sample is equivalent to mean of Model #5 with failure to reject hypothesis at 5% significance level.
**NULL Hypothesis – Variance is equivalent to variance of Model #5 with failure to reject hypothesis at 5% significance level.



(a) (b)

Figure 4.5. Mean of Young's modulus (a) and CTE (b) for variation in element size.

4.2.3.2. Robustness

The robustness of the analysis is tested by studying the statistical distribution of the modulus for two different sample sizes and two different volume sizes. In Figure 4.6, the normal distributions of fifteen and 150 samples are plotted for the converged volume of $R = 50$ and a larger element size for the same volume of $R = 30$. Once again quantitative measures are used to compare standard deviation and mean as shown in Table 3.

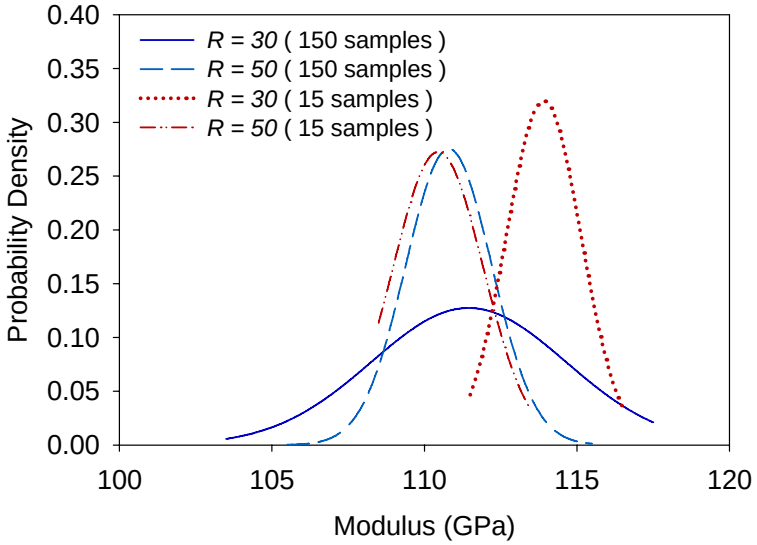


Figure 4.6. Normal distribution of modulus for varying sample sizes.

Table 4.4. T-test and f-test results of Young's modulus for different RVEs at 25ºC.

Model #	Num	R	N	L_{RVE} (μm)	L_{voxel} (μm)	χ (GPa)	S	2 sample t-test*	f-test**
ORNL	15	50	20	12	0.24	110.82	1.71	pass	pass
ORNL	150	50	20	12	0.24	110.65	1.52		
ORNL	15	30	20	12	0.40	113.91	1.26	fail	fail
ORNL	150	30	20	12	0.40	111.15	3.04		
ORNL	150	30	20	12	0.40	111.15	3.04	pass	fail
ORNL	150	50	20	12	0.24	110.65	1.52		
ORNL	15	30	20	12	0.40	111.15	3.04	fail	pass
ORNL	15	50	20	12	0.24	110.65	1.52		

*NULL Hypothesis – Unknown mean is same for both distributions with failure to reject hypothesis at 5% significance level.
**NULL Hypothesis – Unknown variance is same for both distributions with failure to reject hypothesis at 5% significance level.

4.2.4. Final RVE size

4.2.4.1. Modulus RVE

In Figure 4.4, it can be seen that the discretization error controls accuracy while RVE size controls the standard deviation. To gain perspective on variability, the plots are scaled such that the maximum and minimum are approximately twenty percent of the initial mean. Table 4.2 and Table 4.3 support the conclusion drawn from the box plots. The t-test fails for discretization error until a sufficient voxel size is reached, while the standard deviation doesn't vary. However, for RVE size, all t-tests pass for a 95% confidence level. The standard deviations in the chi-square test only pass near the same RVE size.

The modulus results at higher temperatures can be seen to vary more than those at room temperature, especially with regard to RVE size. This is an artifact of equation (4.18) by introducing the variation of CTE in the results for effective modulus.

The box plot data sets look at the variation of results due to changes in R and N. These results also take for granted that the created microstructure is actually significant in determination of material properties. Plus, it is difficult to extrapolate the results of the box plot to broader conclusions about acceptable RVE sizes for voxel reconstructions. Thus Figure 4.5, which plots the mean modulus for realization sets against a ratio of characteristic length to voxel length, is used to draw a broader conclusion;

For $N \geq 15$ *for element sizes less than* λ_{Ni} *the modulus is independent of* L_{RVE} *even if the final result has not converged.*

This indicates that the error is now purely from discretization. At this point, the modulus is entirely dependant on the physical size of each element converging to a consistent point. Once the element size reaches a certain point it can be stated:

For $N \geq 15$ *when voxel length is one half of* λ_{Ni} *the RVE is an accurate representation of the ORNL sample.*

The lineal path realizations also show a good match to experimental values, and since results in Chapter Two and Three showed different connectivity behavior, its accuracy was studied. However, since no analytical expression exists for them and computation time is much larger, they are not necessary to use. The periodic and random realizations have results that serve like upper bounds on the modulus, probably due to the change in pore distribution.

4.2.4.2. CTE RVE

Interestingly, in Figure 4.4(e-f) it can be seen that unlike the modulus which sees a small increase due to insufficient RVE size, the CTE drops with a very large increase in variation. In regards to discretization error a small decrease in value can be seen, but only the largest element size shows significant error.

The trend in CTE is much more linear than that of modulus. Modulus values steadily increase, diverging away from the correct modulus as the element size increases, but even the random distribution provides an approximate measure of CTE for any element size (Figure 4.5(b)). The periodic distribution, however, proves to be completely inaccurate. Therefore, for CTE it can now be stated:

For $N \geq 15$ CTE can be approximated by any distribution with random long range order and the correct volume fractions.

4.2.4.3. Sample size

The previous results assume that five realizations with measurements in three directions are sufficient to describe the microstructure, but it can be shown that the number of samples needed is also dependent on discretization of the realization. In Figure 4.6, as the number of samples increases for the rougher microstructure, the mean shifts toward the more refined data set. For the more refined (and considered converged) data set, the normal distribution barely shifts. These results are apparent in the hypothesis tests listed in Table 4.4**,** and this corresponds with the statement that discretization influences standard deviation.

For CTE the standard deviation for samples is small with values consistently less than one percent of the mean.

4.3. Microstructure variation

4.3.1. Reconstructions

Now that the RVE size is determined, the microstructure can be modified as needed to study porous cermets. To this end, multiple realizations of the three-phase composite were generated using the Debye equation (Chapter 2, eqn. 2.9). The base microstructure set each λ to those determined from analysis of an ORNL sample in Table 2.1. From these base parameters several additional microstructures were generated with either varying porosity or λ, refer to Table 4.5.

Table 4.5. Microstructure realizations.

Mod.	Set	λ_{Ni} (μm)	λ_{pore} (μm)	λ_{YSZ} (μm)	φ_{Ni}	φ_{YSZ}	φ_{pores}	L_{voxel} (μm)	L_{RVE} (μm)	N	R
ORNL	-	0.6	0.4	0.4	0.35	0.43	0.22	0.24	12	20	50
φ_{pores}	1	0.6	0.4	0.4	0.32	0.40	0.28	0.24	12	20	50
	2				0.30	0.36	0.34				
	3				0.27	0.33	0.40				
λ_{pore} λ_{YSZ}	4	0.6	0.6	0.6	0.35	0.43	0.22	0.24	12	20	50
λ_{pore}	5	0.6	0.4	0.6	0.35	0.43	0.22	0.24	12	20	50
	6	0.6	0.8	0.6				0.24	12	15	50
	7	0.6	1.0	0.6				0.23	15	15	65
λ_{YSZ}	8	0.6	0.6	0.4	0.35	0.43	0.22	0.24	12	20	50
	9	0.6	0.6	0.8				0.24	12	15	50
	10	0.6	0.6	1.0				0.23	15	15	65
λ_{Ni}	11	0.4	0.6	0.6	0.35	.43	0.22	0.24	12	20	50
	12	0.8	0.6	0.6				0.24	12	15	50
	13	1.0	0.6	0.6				0.23	15	15	65

4.3.1.1. Porosity

To study the influence of porosity on the cermet, models 1-3 in Table 4.5 were reconstructed with the same characteristic lengths as the ORNL sample, but steadily increasing porosities. As the porosity increased, the ratio between nickel and YSZ is held constant. The results are compared against the ORNL composite sphere model developed by Radovic et al. for Ni-YSZ cermets in Figure 4.7 [25]. Also plotted is the upper bound from the Hashin Sthrikman model, a variational method that calculates bounds based on volume fractions and constituent properties. Lastly the CTE and modulus were plotted against temperature in Figure 4.8(a-b).

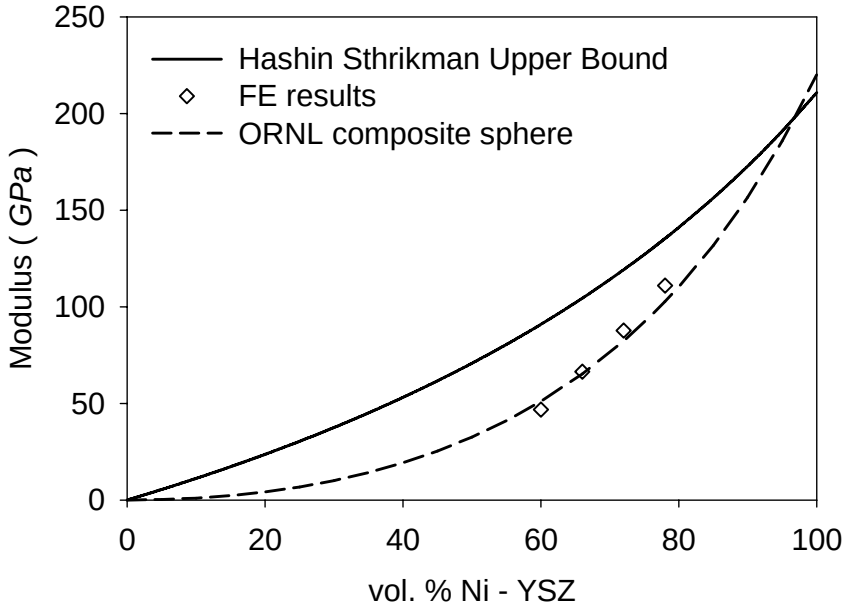
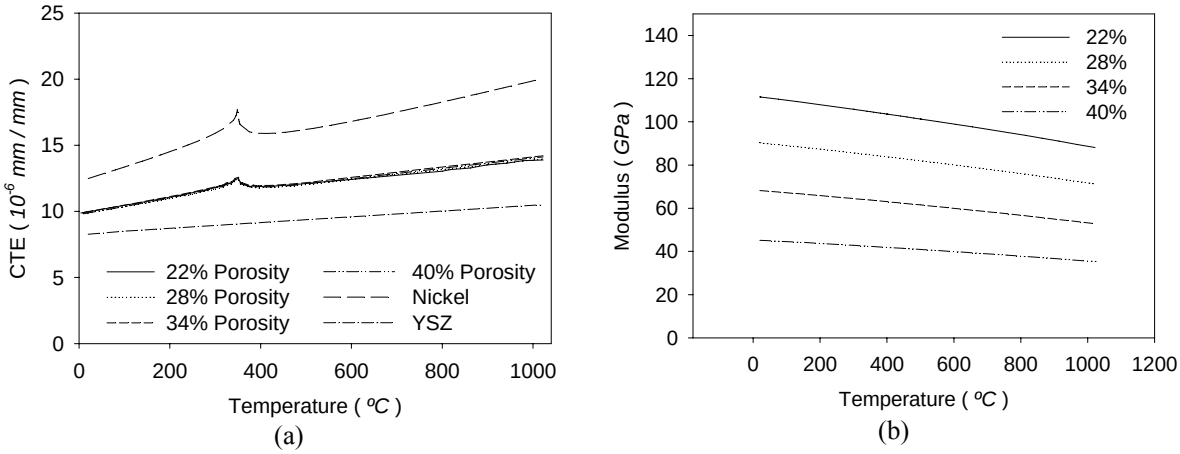


Figure 4.7. Variation of modulus for FE results compared to experimental values.



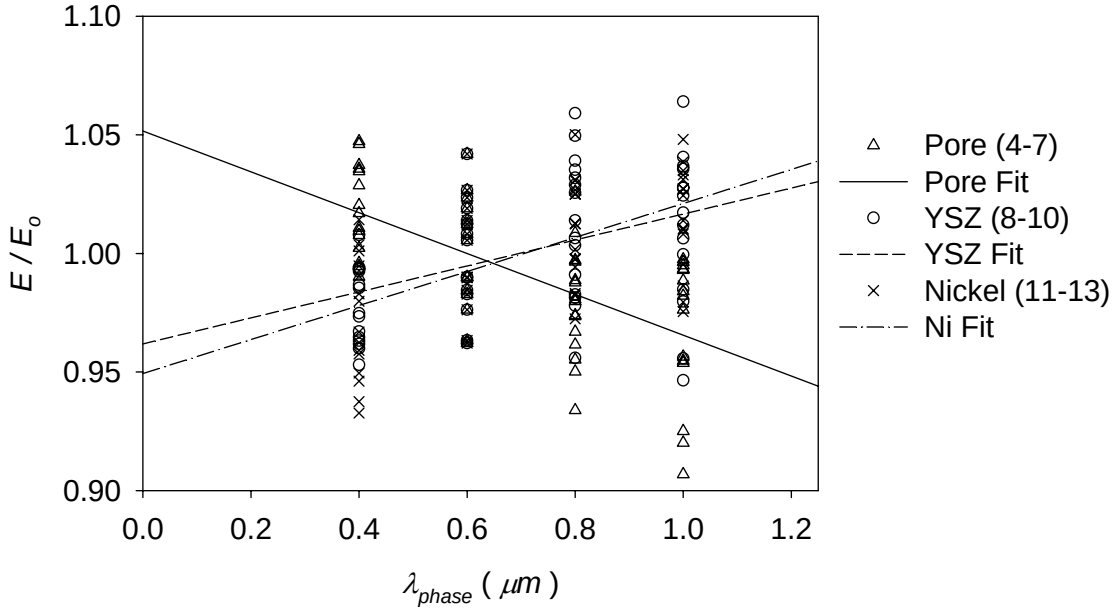
(a) (b)

Figure 4.8. Variation of CTE (a) and modulus (b) versus temperature.

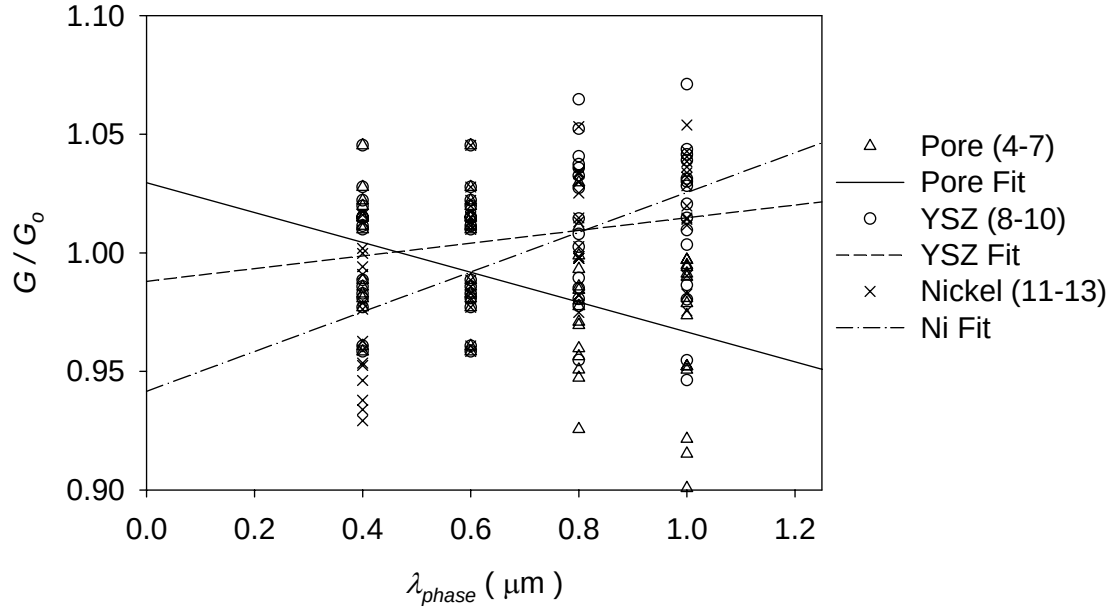
4.3.1.2. Characteristic length

The final Debye functions of the microstructure will be influenced by the starting particle sizes, the sintering temperature and time. So it is reasonable that changing length scale should be studied. In Set 4 the characteristic length was set equal for all three phases, and found to be roughly equivalent to the ORNL sample. Set 4 is now used as the base for changes in length scale. Three different cases are examined where λ varies for porosity (models 5-7), YSZ (8-10), and nickel (11-13). The volume fractions are held constant. The results from Section 4.2 were used to maintain acceptable R and N values. However, this does mean the volume and voxel size may vary across sets.

For each case, five models were created, and the modulus, shear modulus, and CTE were determined in three directions for fifteen samples. In Figure 4.9 and Figure 4.10 these properties are plotted against the changing λ for each phase and is normalized by the mean values from Model 4, designated E_o, G_o, and CTE_o. For each data set, a linear fit is also shown.



(a)



(b)

Figure 4.9. Young's modulus (a) and shear modulus (b) against λ for one phase while with other phases are at 0.6 μm.

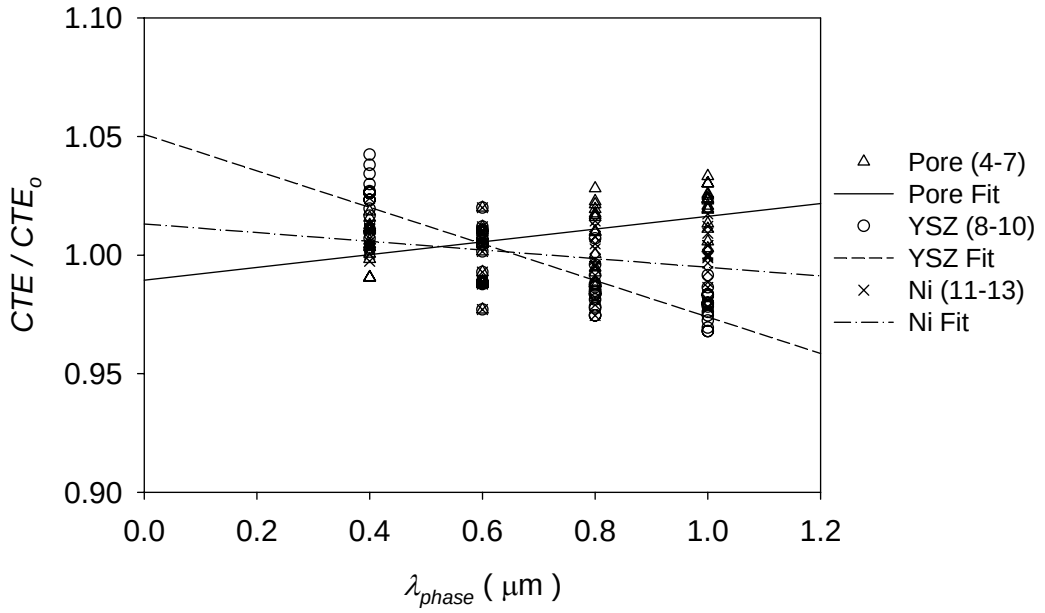


Figure 4.10. CTE against λ for one phase while other phases are at 0.6 μm.

4.3.2. Microstructural impact

Varying the characteristic lengths changed the resulting properties less than five percent, while changing porosity brought about a significant change in modulus and had no impact on CTE. Therefore, the cluster function and the parameters determined in Chapter 3 were used to provide further insight into microstructural changes.

4.3.2.1. Porosity

In Figure 4.7 the predicted modulus of the Debye realizations show a good match to previous experimental results, and it can be seen that as the porosity increases the modulus decreases in a nonlinear manner. After temperature is added, additional stresses are generated, since now nickel and YSZ also undergo stresses due to thermal mismatch. This explains why, in Figure 4.8, CTE does not change with volume fraction and is in

fact dependant on the ratio between the volume fraction of nickel and YSZ. It also shows that outside the Curie point region in nickel, the thermal expansion is primarily linear. Both plots in Figure 4.8 are reasonable in reference to experimental results [25, 27].

Due to the calculation of modulus from equation (4.18) and the linear properties used for nickel's modulus, the effective modulus is linearly temperature dependent. However, as the amount of nickel drops, the variation of the modulus due to temperature continues to slightly decrease. This is seen in the steadily decreasing slopes of each curve where at 22% porosity the modulus drops by 0.023 GPa/ºC and at 40% porosity it changes by .010 GPa/ºC.

Theory holds that the percolation threshold should be related to the effective modulus through a power law relation such that

$$E \propto \left(\varphi_i - \varphi_{ic}\right)^{\beta} . \tag{4.20}$$

β is the power law exponent and φ_{ic} is the percolation threshold. Feng et al. found that for the Swiss cheese model, the modulus exponent was equal to 5/2 [69]. When (4.20) is applied across different porosities, a value for β of -2.85 ± 0.056 at room temperature is found.

4.3.2.2. Internal length scales

There is consistently an inverse relationship between changes in length scales for porosity and the other phases. As porosity increases, the modulus steadily drops, although the length scale would need to be fairly large before even a five percent change would occur. For CTE, the behavior is flipped and now an increase in pore size increases CTE. Another parameter is needed to relate change to one phase for all variations of the internal length scales.

As discussed in Chapter Three, the percolation threshold can be approximated from the point when the cluster function reaches its long term values. In Table 4.6 it can be seen that these percolation values stay approximately stable despite changes in λ.

Table 4.6. Percolation threshold for different microstructures.

λ (μm)		**Percolation Threshold**		
Ni=YSZ	**Pore**	**Ni**	**YSZ**	**Pore**
0.6	0.40	0.120	0.180	0.040
0.6	0.60	0.115	0.184	0.038
0.6	0.80	0.114	0.178	0.034
0.6	1.00	0.112	0.176	0.043
Ni=Pore	**YSZ**			
0.6	0.40	0.121	0.175	0.038
0.6	0.60	0.115	0.184	0.038
0.6	0.80	0.114	0.198	0.042
0.6	1.00	0.116	0.212	0.045
YSZ=Pore	**Ni**			
0.6	0.40	0.111	0.188	0.038
0.6	0.60	0.115	0.184	0.038
0.6	0.80	0.130	0.178	0.040
0.6	1.00	0.140	0.180	0.044

Next the point, L_c^{pore}, introduced as the distance at which the percolation threshold is reached, is examined. Figure 4.11 and Figure 4.12 compare L_C^{pore} for all the models of

varying microstructures and 22% porosity (models 4 – 13 in Table 4.5). When the effective properties are correlated to L_C, a definite link can be drawn between the clustering of porosity and the resulting modulus and CTE for all variations in λ.

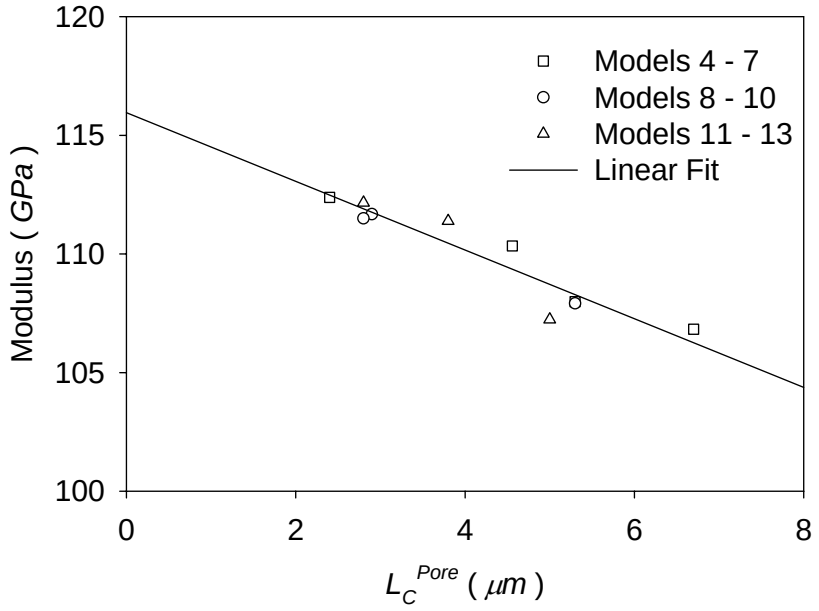


Figure 4.11. Change in length at percolation threshold of porosity for modulus.

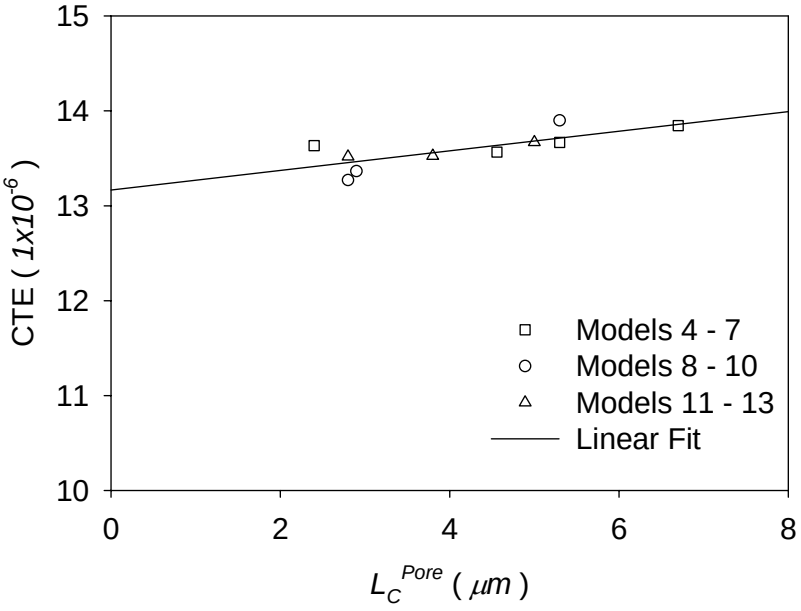


Figure 4.12. Change in length at percolation threshold of porosity for CTE.

4.4. Summary

The three objectives during the determination of RVE size for structural properties were to obtain a rule of thumb, verify accuracy, and verify robustness of the analysis. It was found that an $R = 50$ and $N = 20$ is more than sufficient to capture material behavior, plus a minimum of $R = 40$ and $N = 15$ will work for both properties. This applies to displacement boundary conditions and any edge effects resulting from them. Upon examining Figure 4.8 it can be stated that for three-phase Debye media the modulus needs an element size half of the characteristic length and, the CTE will vary little depending primarily on ratio between Ni and YSZ in a distribution with random long range order.

Both the analysis of robustness and comparison to experimental results show that the voxel reconstruction works for the analysis.

When the voxel reconstructions were used to predict changes in structural properties due to microstructural details, it was found the most significant change was due to change in volume fraction of porosity. The importance of percolation of the pore phase also would explain why the lineal path and two point reconstructions has the same modulus. Despite differences in the nickel and YSZ phases, the pore phases were exact (recall Table 3.1). Overall, the behaviors of the porous cermet can be summarized as follows:

- modulus is most influence by changes in volume fraction,
- CTE is dependent on the ratio of the volume fractions of nickel to YSZ and not the volume fraction of pores,
- at 40% porosity the impact of temperature on modulus is minimal,
- an increase in modulus will result in a decrease of CTE and vice versa, and
- finally pore percolation can be correlated with modulus and CTE.

The next chapter moves from a structural analysis to a transport analysis to determine the effective thermal conductivity of the porous cermet.

CHAPTER 5

EFFECTIVE THERMAL CONDUCTIVITY

Previously, a methodology to study the structural properties of porous cermets was introduced, but a slightly different formulation is needed to study transport properties, like thermal conductivity. Transport properties differentiate from a structural analysis due to the occurrence of a direction-dependent flux field. For thermal conductivity, this is an energy field that correlates to a temperature change in any given direction. This chapter will investigate the voxel reconstruction's ability to determine transport properties for the cermet.

As in the previous chapter, the first step examines the variation in conductivity with changes in RVE size and discretization of the microstructure. However, an FE model with perfect bonding between elements is not capable of accurately predicting a composite's thermal conductivity, and another factor must be taken into account. This factor is energy loss due to the interfacial resistance between nickel and YSZ. Since this resistance is dependent on morphology, surface roughness, and temperature variation, it is not realistically possible to measure an accurate value that can be easily used in analysis of the anode. To that end, a simple theoretical model is used alongside published experimental data to modify the FE results. A numerically determined thermal resistivity is then used to study the change in conductivity for different microstructures.

5.1. Thermal resistance model

For any material, the heat flux and temperature change can be related through thermal conductivity, κ, using Fourier's Law. Equation (5.1) is the one-dimensional form of this law. Thermal conductivity then is an inherent material property that describes a material's ability to transfer heat. Equation (5.1) relates the change of temperature, ΔT, in a given direction for a length, Δx, to the heat flux, $\dot{Q}$, such that

$$\dot{Q} = -\kappa A \frac{\Delta T}{\Delta x}. \tag{5.1}$$

The area, A, is perpendicular the heat flux and normal vector, $\tilde{n}$, as illustrated in Figure 5.1.

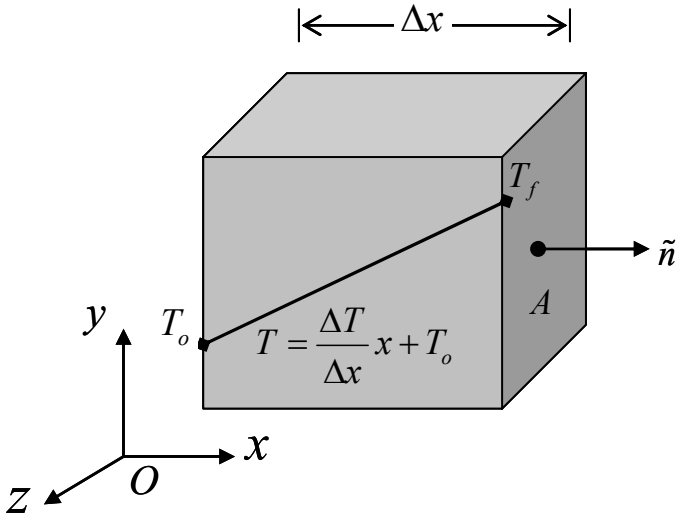


Figure 5.1. RVE in transport model.

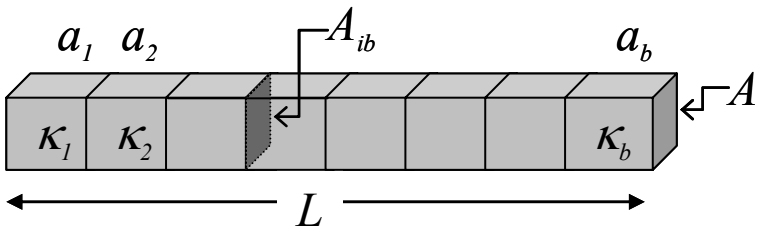


Figure 5.2. Illustration of thermal conductivity model.

In a composite material, the effective conductivity will be related to the bulk constituent properties and also to the behavior at the interfaces of those constituents. For example, assume that a composite material is made of a B length of blocks, where each block has a b thermal conductivity and length, as illustrated in Figure 5.2. The total length of the bar is the sum of each individual block length, a function, a_b, as illustrated in Figure 5.2, so that

$$L = \sum_{b=1}^{B} a_b = B \bullet \langle a_b \rangle = Ba \,, \tag{5.2}$$

where $\langle a_b \rangle = a$ is the average length of the blocks.

Furthermore, each b block of a material will have a thermal conductivity, κ_b. Thus the resistivity of the nth block is

$$r_b = 1/\kappa_b \,, \tag{5.3}$$

and the resistance of the nth block is now a function of geometry and resistivity such that

$$R_b = \frac{r_b a_b}{A} \,. \tag{5.4}$$

At the interface an interfacial resistivity, r_{ib}, is used to find the total interfacial resistance, R_{ib}, for $B-1$ interfaces, and a total length, L.

$$R_{in} = \frac{L}{A_{ib}} r_{ib} . \tag{5.5}$$

In equation (5.5) A_{ib} is the total of the interfacial area normal to the direction of heat change across the entire length. It should be noted that for the actual cermet material, the solid to pore interfaces are neglected in this formulation.

A factor, ρ, can now be written as

$$\rho = \frac{L}{A_{ib}} . \tag{5.6}$$

The parameter ρ lends well to use with digitized microstructures such as the voxel reconstructions. The interface area is determined simply by counting the number of connecting blocks of nickel and YSZ in the desired direction.

Going back to Figure 5.2, the total resistance along the entire bar is

$$R_{total} = \sum_{b=1}^{B} R_b + R_{ib} , \tag{5.7}$$

and the total heat flux of the composite can now be written as

$$\dot{Q} = -\frac{\Delta T}{R_{total}} . \tag{5.8}$$

Replacing Δx with L for the entire length of the bar and substituting in equation (5.1), an expression for thermal conductivity can be found, such that

$$\kappa = \frac{L}{R_{total} A} . \tag{5.9}$$

Substituting (5.7) into (5.9) the thermal conductivity is now

$$\kappa = \frac{1}{\sum_{b=1}^{B} \frac{r_b a_b}{L} + \rho r_{ib}} . \tag{5.10}$$

Note that a_b/L is equal to the volume fraction of block n; thus the finite element solution without considering interfaces can be set equal to

$$\frac{1}{\kappa_{FEM}} \approx \sum_{b=1}^{B} r_b \varphi_b \ . \tag{5.11}$$

Equation (5.10) can now be rewritten as

$$\kappa = \frac{1}{\kappa_{FEM}^{-1} + \rho r_{ib}} \ . \tag{5.12}$$

To use (5.12), consider a microstructure that has the experimental thermal conductivity, $\kappa_{\exp}^{(o)}$, i.e.

$$\kappa_{\exp}^{(o)} = \frac{1}{\left(\kappa_{FEM}^{(o)}\right)^{-1} + \left(\rho^{(o)}\right) r_{ib}} \ . \tag{5.13}$$

In equation (5.13) the superscript (o) indicates that these quantities are related to a particular microstructure.

Solving for interfacial resistance

$$r_{ib} = \frac{1}{\rho^{(o)}} \left(\frac{\kappa_{FEM}^{(o)} - \kappa_{\exp}^{(o)}}{\kappa_{FEM}^{(o)} \kappa_{\exp}^{(o)}} \right). \tag{5.14}$$

Note that r_{ib} is independent of microstructure, while $\rho^{(o)}$ is determined from the specific reconstruction. This means that once r_{ib} is determined, the thermal conductivity can be determined for any realization using (5.12). As mentioned the previous formulation works well for the discretized microstructure; however, it does not necessarily determine the actual interfacial resistivity of nickel and YSZ. This is mainly due to the simplified approximation of the microstructure.

5.2. FE model

At room temperature, the structural properties of nickel and YSZ are very similar; however, the temperature-related properties of nickel are both significantly larger and more variable than YSZ. Table 5.1 lists the relevant material properties, taken from the literature, at room temperature and at 1000ºC [27, 92, 93]. A complete list of the temperature dependent properties can be found in Appendix D. Experimental studies of the electrolyte material, non-porous nickel 8mol% ytrria (YSZ) material, were used as the YSZ properties in the Ni-YSZ composite [27, 33, 87]. Nickel is treated as a general polycrystalline material with primarily linearly dependant temperature properties except near the Curie point. The CTE and specific heat of nickel specifically show an exponential jump in value around the Curie point due to the paramagnetic transition. Nickel's thermal conductivity also varies significantly around the Curie point, but with an inverse relationship to that of CTE and specific heat [94].

The pore elements in the sample are set equal to an argon atmosphere. It should be noted that at room temperature, the thermal conductivity of nickel is approximately 43 times that of YSZ and 4844 times that of argon. Temperature dependence was neglected for argon.

Table 5.1. Comparison of constituent material properties.

Property	Temperature (ºC)	Ni	YSZ	Argon
κ(W / (mºC)	127	77.5	1.89	.016
	1000	70.0	2.05	.016

5.3. Results

5.3.1. RVE Size

The methodology for determining RVE size was the same as that used by Johnson and Qu, which first finds the acceptable element size by varying R and then finds the acceptable RVE size by varying N [36]. Recall that R is equivalent to B from Figure 5.2. Box plots of these results are shown in Figure 2. For each set of five samples, the thermal conductivity was measured in three directions providing a total of fifteen data points and also confirming that the effective value is independent of direction. To determine that a sufficient number of samples were collected, histograms were plotted of the data spread for 15 samples and 150 samples at an element size before and after convergence (refer to Figure 5.4). This is done for the FE results without accounting for interfacial resistance.

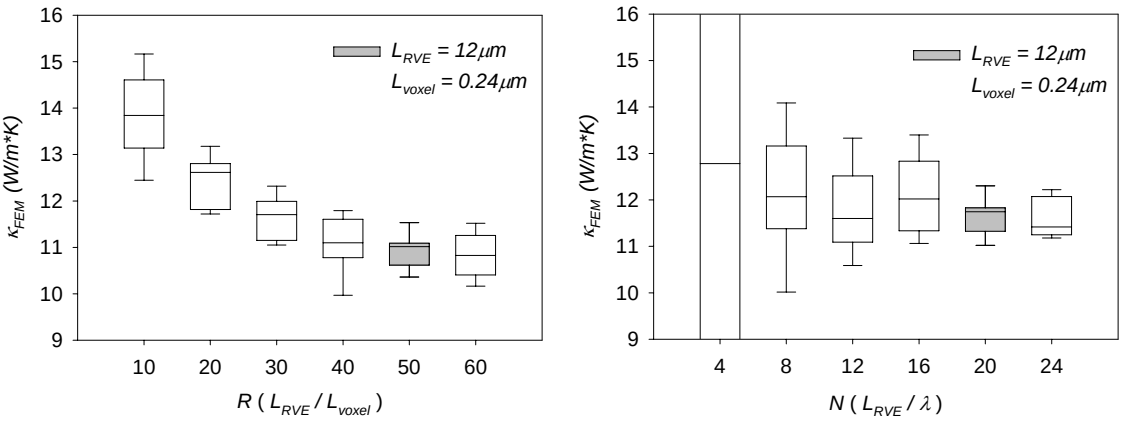


Figure 5.3. Box plots of discretization error and RVE size, respectively, for Young's and the uncorrected thermal conductivity (a-b).

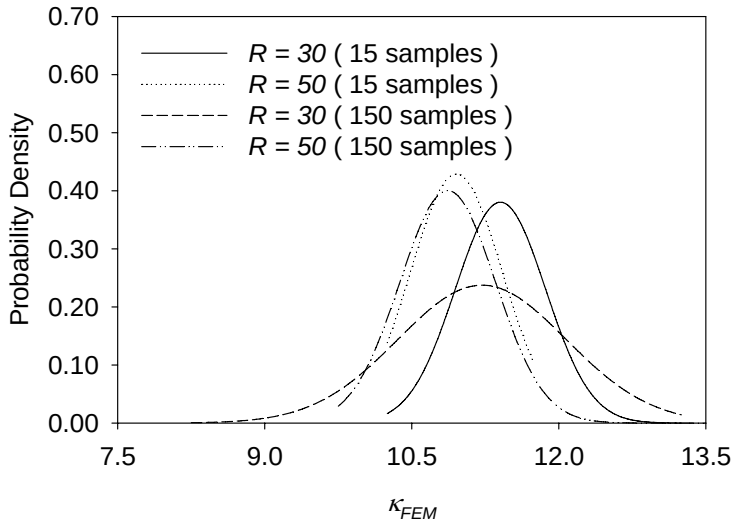


Figure 5.4. Histogram of FE thermal conductivity for different sample sizes and elements sizes at $N = 20$.

5.3.2. Interfacial Resistance

Equations (5.12) and (5.14) were used along with experimental results for Ni-YSZ with 34% porosity from work by Radovic et al. [27] to determine the interfacial resistivity as defined by (5.12). The FE results used to determine resistivity were obtained from ORNL realization set where N was equal to 20 and R was equal to 50. The interfacial resistivity results are plotted against temperature in Figure 5.5. In Figure 5.6 the original FE thermal conductivity is compared to the experimental results from ORNL, and the FE results are seen to be as much as 50% higher than experimental values.

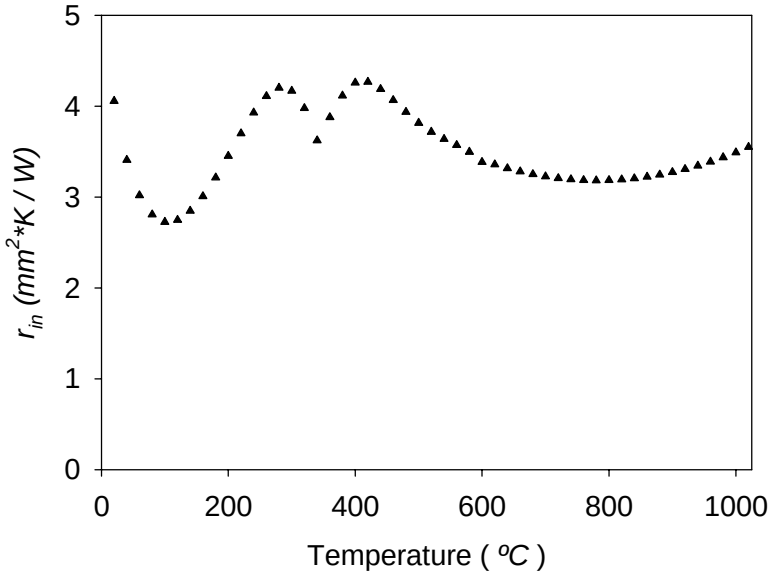


Figure 5.5. Interfacial resistivity determined at 34% porosity.

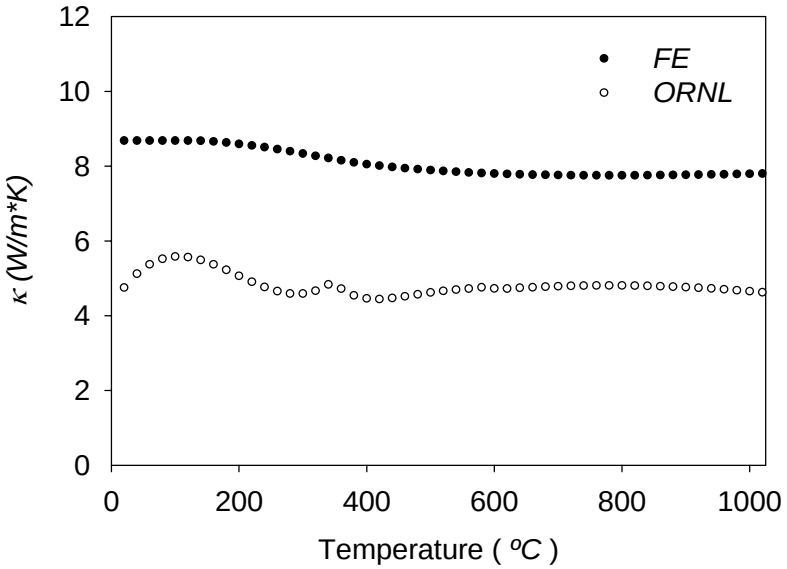


Figure 5.6. FE results without interfacial resistance.

5.3.3. Varying microstructures

The interfacial resistivity from Figure 5.5 was used to plot the thermal conductivity for 22%, 28%, and 40% porosity, shown in Figure 5.7a. The results are compared to ORNL thermal conductivity values (Figure 5.1b) at different porosities and are found to match well. The characteristic length of the microstructure for the different porosities was left equal to the original ORNL sample (refer Table 2.1). The next set of realizations assumed that nickel and YSZ both had a λ of 0.6 μm, while the length scale of the pore phase was allowed to vary. These are model numbers 5-7 listed in Table 4.5. The thermal conductivity results for these realizations are plotted Figure 5.8. The results are averaged over fifteen samples.

For each new realization set, the values that change are the FE results and the $\rho^{(o)}$ value, while r_{ib} remains the same. In Table 5.2 and Table 5.3, the ratio of the nickel to YSZ interface to the total area is recorded. In Table 5.2 the results were found to be linearly related to the porosity with a negative slope of 28 percent and an intercept of 0.17.

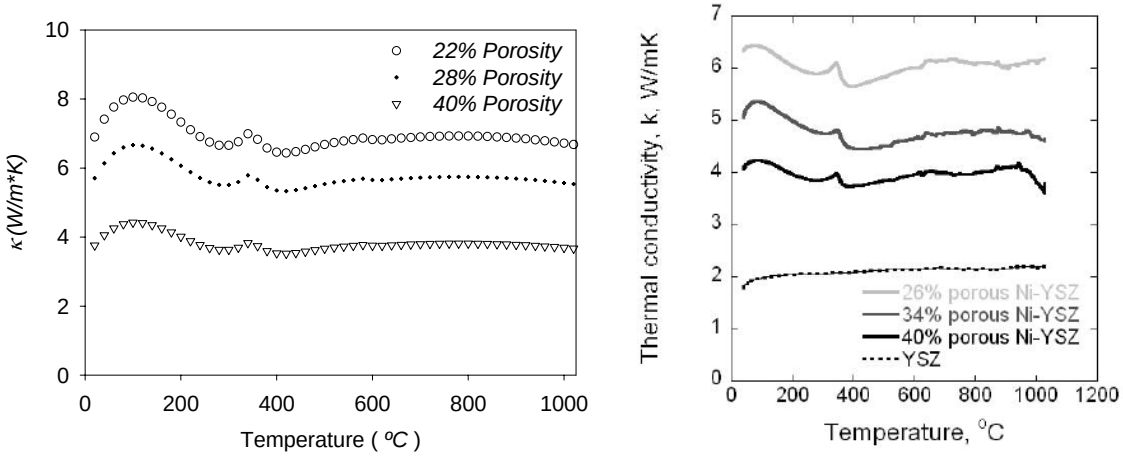


Figure 5.7. Corrected thermal conductivity versus porosity for numerical analysis (a) and ORNL results [27] (b).

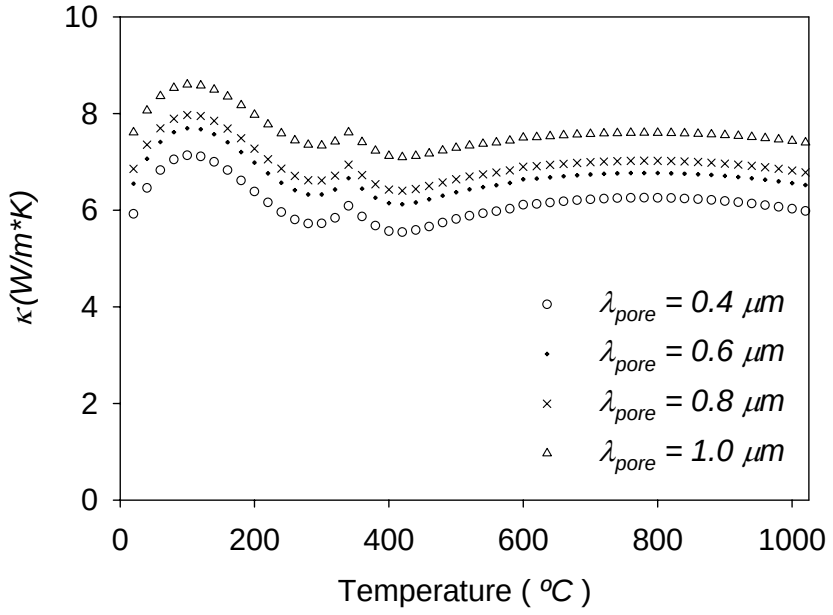


Figure 5.8. Thermal conductivity for changing length scales in pore phase.

Table 5.2. Interface area and conductivity at RT for changing porosity.

φ_{pore}	**.22**	**.28**	**.34**	**.40**
A_{in} / A	.107	.088	.071	.056
κ $((W/(m \bullet K))$	6.89	5.70	4.75	3.76

Table 5.3. Interface area for changing length scales.

$\lambda_{pore}\ (\mu m)$	**0.4**	**0.6**	**0.8**	**1.0**
A_{in} / A	.077	.098	.109	.113
κ $((W/(m \bullet K))$	5.92	6.55	6.85	7.61

5.4. Discussion

5.4.1. RVE size and discretization

Examination of Figure 5.3 leads to the same conclusion for thermal conductivity as for structural properties, i.e. discretization of the microstructure controls accuracy while the RVE size controls standard deviation. However, thermal conductivity seems to have a much higher variability overall. The minimum acceptable R and N are 50 and 20, respectively. This is compared to 40 and 16 for the modulus. Plus, in Figure 5.4(a), the shift of the less accurate model to a correct mean is much more pronounced, suggesting that the model itself is fundamentally wrong at this lower discretization. The higher variability in the thermal conductivity most likely results from the addition of

direction in the formulation. The effective conductivity is now dependent on the variability of the interfaces in one direction as the temperature changes.

5.4.2. Numerical results

The addition of interfacial resistivity had a significant impact on the FE predictions of thermal conductivity, which dropped by as much as 50%. In Figure 5.7 the predictions of thermal conductivity for different porosities match well to experimental results [27]. Knowledge of the accuracy of the thermal conductivity values for different length scales would require additional experimental results, but there is an obvious increase in thermal conductivity with an increase in interfacial area. The reason is hard to verify, but a reasonable assumption is increasing Ni-YSZ interfaces means a drop in argon to solid interfaces.

The final point of interest is the change in interfacial area for both volume fractions and length scales. First it should be noted that in a digitized medium, the slope of the two-point probability function can be related to the resulting surface area of a phase, s, as shown in equation (5.15) [63].

$$\left.\frac{d}{dr}S_2(r)\right|_{r=0} = -\frac{s}{2d}, \text{ where} \tag{5.16}$$

in (5.16) the variable d refers to the dimensionality of the system. This relationship between surface area and 2-point probability functions explains the linear change in interfacial area for varying volume fractions. The slopes of the probability functions do not vary with increases in volume fraction. However, these slopes will change with length scale. This is significant since thermal conductivity now varies linearly with

porosity, but shows no obvious correlation to length scales. This behavior is the reverse of how the modulus behaves with changes in length scales and volume fractions.

5.5. Summary

Although the finite element solution alone does not accurately predict the thermal conductivity for different microstructures, the addition of interfacial resistivity appears sufficient to predict thermal conductivity. The use of interfacial area to link the thermal conductivity values also provided insights into how the property will vary with microstructure. It is significant that a change in porosity results in a linear change of thermal conductivity as compared to the results for modulus and CTE in the previous chapter.

One issue with the transport analysis is its dependence on experimental data and the constituent properties used in the analysis. This can especially be seen in the Curie point behavior, which despite being included in the FE model, does not carry through to the FE results, especially compared to published values that use the specific heat and Lorenz number to calculate the thermal conductivity [27]. However, an extensive study of the Curie point behavior is outside the scope of this work.

For both Chapter Four and Five effective properties were determined for linear material behaviors (except at the Curie Point). The next few chapters will investigate nonlinear material behaviors.

CHAPTER 6

DAMAGE AND PLASTICITY

In fuel cells, the anode will not undergo a typical loading scenario and it can be expected that manifold constraints, sealing at edges, and contact with interconnects could result in areas of high localized stresses leading to failure in sections of the anode. Additionally, in anode supported stacks, the anode bears the majority of the load, having heights up to 1mm, compared to the electrolyte and cathode that have heights less than 20 μm. Thermal mismatch between the electrolyte and anode layers during the tape cast process is yet another factor that will induce stresses in the anode. It is possible that during pSOFC construction, the initial warparge of the PEN from sintering and then the process of assembly in the stack could lead to localized failures in the anode, even before operation.

For these reasons, the study of damage and plasticity in Ni-YSZ is significant to fuel cell behavior, but the cermet failure is also useful to the overall study of composites. In the cermet, two contrasting material behaviors occur around a continuous distribution of pores. In brittle YSZ, failure can be assumed to be instantaneous and localized, while the plastic nature and lower strength of nickel suggests a more distributed loading. How these behaviors govern the bulk response of the composite is not easily discerned without a three-dimensional numerical analysis. The significance of microstructural distribution on bulk behavior is even harder to quantify.

Finite element methods have often been used to study damage and plasticity in composites. Lee, et al. incorporated damage into a multi-scale tessellation FE model, using elliptical cracks, which were allowed within each particle [39]. The work was extended to incorporate damage and plasticity by Ghosh, Lee, et al. in a review of VC-FEM in 2001 [41]. Ghosh also tracked damage evolution in subsequent works [95]. The combined works of Segurado and Gonzalez studied the effect of clustering on the total strength of metal-matrix composites with a nonlinear FE analysis [79, 96].

Kumar et al. used the simulated annealing method to perform an elastic-plastic analysis on multi-phase composites [83]. Mishnaevsky et al. modeled damage growth and fracture by either "softening" or disappearing elements as a given criterion is exceeded [84, 86]. Next, Mishnaevsky used the Rice-Tracey damage parameter to measure void growth in porous and graded composites [85]. Cannillo and Carter studied realistic and idealized brittle materials using an FE analysis combined with a Weibull probability criterion for individual element failure [52]. Polycrystalline structures were studied in a similar way by Zimmerman et al. by applying the Griffith failure criteria to grain boundary elements [97]. Singh et al. used the maximum principal stress criterion and yield stress to study damage and plasticity in discontinuous reinforced aluminum alloys [50]. To study fiber composites, the McClintock void growth model was used [47].

Once the microstructure is analyzed, there are a variety of methods to relate the material to the microstructure. FE contour plots provide a visual representation of stresses and strains, but histograms of stress distributions, cumulative probability plots, and standard deviations of field data can quantify the results [79, 82]. Other work has

focused on finding microstructural "hot spots" or the creation of behavior-related correlation functions [83]. Kumar et al. used an elastic-plastic analysis FE model to compare stress-strain curves, stress histograms, and shear band localization for multiple realizations [83].

In the determination of effective properties (refer to Chapters 4-5), the focus was on determining the mean and statistical distribution of common material properties for an acceptable discretization and RVE size. In the study of nonlinear behavior, there was a brief look at the trend in convergence of the sample size and the different measures of damage and plasticity. Then, the focus shifted to the impact of damage and plasticity on bulk behavior and its correlation to the probabilistic features of the microstructure.

Stress-strain curves were used to provide a bulk description of the material behavior, but different measures were needed to understand the interaction between phases. Three primary methods were used: phase decomposition, distribution fits, and finally, the mark probability function. Through these methods, the following features of porous cermets were investigated:

- the internal stress distribution in each phase,
- the size and distribution of failure in the microstructure, and
- the significant features that influence the bulk response.

In the following sections, the mark probability function will be introduced along with a brief review of the use of volume averages in the RVE analysis. The material behaviors for nickel and YSZ will be described and the procedures for data analysis briefly discussed. It should be noted that the analysis is based on the small deformation assumption, and cannot accurately model large scale deformations. In determination of the model size, a look will be given to modulus, and yield stress and stress-strain curves

will be examined. The rest of the chapter will focus on the internal stress distribution and failure behavior of one Ni-YSZ realization. Changes in the behavior of the porous cermet will be studied against porosity and probability functions.

6.1. Theory

The realizations were loaded with kinematic boundary conditions using the same as in Chapter Four. A displacement is applied along the boundary; in this case the displacement changes with time, such that

$$u_i = C_1(t) \text{ for } x_i = L_{RVE} \text{ and} \tag{6.1}$$

$$u_i = 0 \text{ for } x_i = 0 \text{ for } i = 1,\ 2, \text{ or } 3\,. \tag{6.2}$$

As the displacement increases, individual elements in the nickel and YSZ phases will behave depending on their specific failure criteria. The slow failure of individual elements will change the stress-strain relationship from linear to nonlinear. Since this point occurs at different times for different realizations, a consistent criterion to determine nonlinearity is needed. This condition is met through the 2% yield offset, which finds the yield stress at the intersection of the stress-strain curve and the linear offset of the modulus.

Recall that the volume average of stresses, designated with $\langle \sigma_{ii} \rangle$, can be decomposed into the average stresses occurring in each phase,

$$\langle \sigma_{ii} \rangle = \sum_{p=1}^{n} \left[\varphi_p \langle \sigma_{ii}^p \rangle \right]. \tag{6.3}$$

In equation (6.3) p designates phase number for n total phases, while the double i subscripts refer to the direction.

Equation (6.3) provides a relationship between the stresses and the volume fraction of each phase. Strain is another matter, being volume dependent; it is a sum of the total strains occurring in the microstructure.

Currently lacking is a means to connect the actual microstructural behavior to the stresses and strains occurring in the microstructure. To that end, the mark probability function is used as introduced by Pyrz [98].

First, recall that the two-point probability function, $S_2^{(ij)}$, can be used to completely describe any phase in the microstructure by determining the probability that any two points will lie in the same phase. Next, define a mark, $m_2^{(b)}$, which is any field behavior that meets the conditions of an arbitrary binning process, b. Binning can be limited to one phase or combination of phases, magnitude of a quantity, and so on. Now the mark function, $M_2^{(b)}$, combines the user-defined mark with the microstructure's probability functions such that

$$M_2^{(b)} = \frac{m_2^{(b)}(\tilde{x}_k, \tilde{x}_l)}{S_2^{(ij)}(r)}. \tag{6.4}$$

Here the $M_2^{(b)}$ function becomes a function of the orientation of two different points, $\tilde{x}_k$ and $\tilde{x}_l$, since the mark is not only dependent on the microstructure but also on the loading conditions that result in the field behavior.

Equation (6.4) is normalized by the two-point probability function. It is designated with ij to allow for normalization by one phase, a combination of phases, or all phases.

For the isotropic homogenous media, the mark function will have bounds that can be defined as

$$M_2^{(b)}(0) = \frac{m_2^{(b)}(0)}{\varphi_{ij}}. \tag{6.5}$$

As r approaches zero, equation (6.5) is the probability that any given point in the phase, or combination of phases ij, will satisfy the conditions of $m^{(b)}$.

6.2. Methodology

6.2.1. Finite element model

As before, each voxel is treated as a material point and perfect bonding is assumed to occur between the elements. Damage and plasticity are incorporated in the FE model through standard material models from the software, Abaqus 6.8-1 [99]. In Appendix D these property files are listed, and they are further described in 6.2.2. Pore elements are deleted to improve computation time. Artificial damping is also implemented to enable convergence of the FE solution, but no other artificial methods are used. The stabilization parameter was optimized using the largest FE models, and once determined, kept constant for all models.

6.2.2. Constituent properties

6.2.2.1. Damage

Radovic et al. reported biaxial strengths of 345 MPa and 209 MPa for room temperature and 800ºC, respectively, in 8mol% YSZ [32]. For this work, the ultimate uniaxial tensile strength was conservatively set equal to these biaxial strengths, with a

linear change between the two temperatures. The maximum stress criterion was such that failure occurs when

$$\max\left|\sigma_1, \sigma_2, \sigma_2\right| \geq \sigma_{ult} . \tag{6.6}$$

Before this point, the material is elastic and isotropic. Failure does not occur in compression. Once failure occurs, damage occurs in the direction of the principle stress.

The Abaqus option for tension stiffening was added to the solution, with the addition of a damage parameter as shown in equations (6.7) and (6.8), so that

$$\sigma = \sigma_{ult} \exp\left(-\varepsilon_p\right) \text{ for } \varepsilon_p \geq 0 \text{ and} \tag{6.7}$$

$$D = 1 - \frac{\sigma}{\sigma_{ult}} . \tag{6.8}$$

In equation (6.7) ε_p is plastic strain and σ_{ult} is the ultimate strength. The damage parameter has no impact on the FE analysis and is only used as a measure of failure for a given FE element.

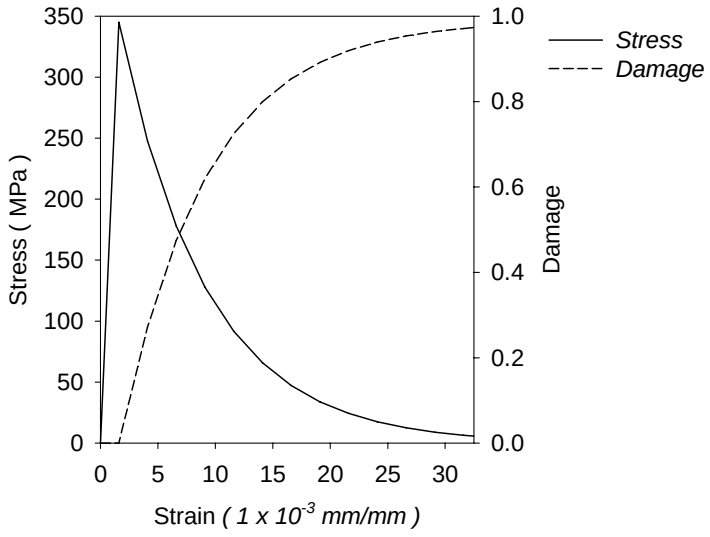


Figure 6.1. Stress-strain in the YSZ element.

6.2.2.2. Plasticity

The plastic stress-strain curves are input as data points for four different temperature loadings. These data points were determined using the Chakrabarty law such that stress is a piecewise function as shown in equation (1.9).

$$\sigma = \begin{cases} E\varepsilon, & \sigma \le \sigma_y \\ \sigma_y \left(\dfrac{E\varepsilon}{\sigma_y} \right)^n, & \sigma > \sigma_y \end{cases}, \tag{1.9}$$

where n is a hardening parameter.

Thompson found that yielding in nickel is a function of grain size, and the Hall-Petch relation was used to determine the yield stress for a grain size of 1μm [100, 101]. A slight temperature dependence in the yield stress and the hardening exponents in equation (1.9) were assumed due to experimental results from Srinivas et al. [102].

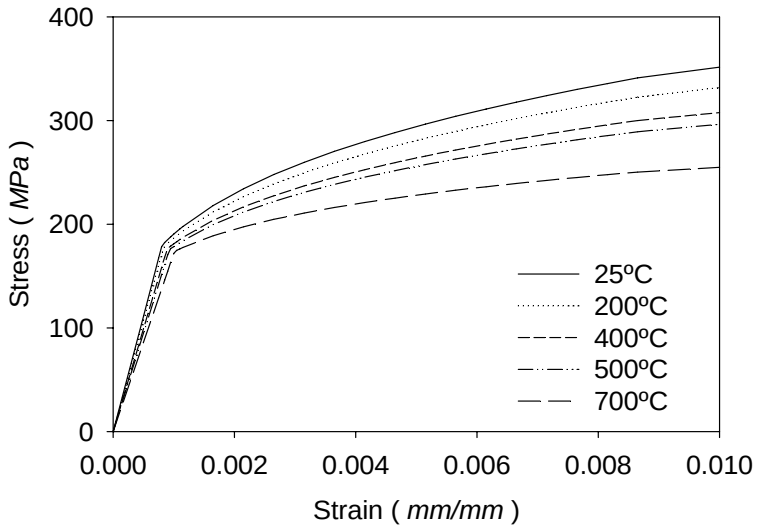


Figure 6.2. Plastic strain curves for nickel.

The classical metal plasticity law is linearly elastic before failure. Isotropic hardening is used and yielding occurs when the von Mises stress reaches the yield stress.

6.2.3. Data analysis

As mentioned in the introduction, three techniques are used to study the field behaviors occurring throughout the microstructure. These are stress decomposition, distribution fitting, and finally, the mark function as described in Section 6.1. These tools treat each element, or voxel, as its own spot in the continuum media. The value at the centroid of each element is found by interpolating from the nodal values. Using the element values, the data can be averaged, input into histograms for data fitting, or binned for use in a mark calculation. The output histograms are fitted to a 3 parameter Gaussian distribution, a 4 parameter Weibull distribution, or left as a histogram. The distribution fitting tool used was the commercial software SigmaPlot and specific equations are provided in Appendix C [70].

6.2.4. Microstructure analysis

The extensive and detailed analysis of multiple realizations in Chapter Four is neither computationally realistic nor necessarily relevant to the damage and plasticity analysis. The priority is to link the behavior to specifics of the microstructure, and to determine worst case scenarios for failure in Ni-YSZ. Therefore, the analysis focuses on a 40% porous microstructure with the same distribution as the ORNL sample. To study convergence behavior, 4-5 samples were analyzed, where each model was strained in the x-direction via displacement boundary conditions. For modified microstructures, one to three samples were deemed sufficient. The models examined are listed in Table 6.1.

Table 6.1. Damage and plasticity microstructure realizations.

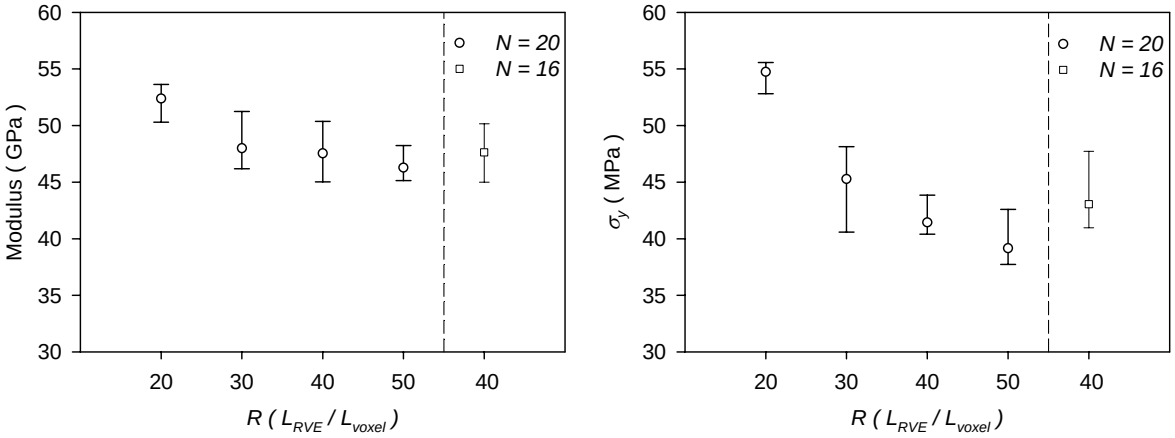
Mod.	**Set**	**#**	λ_{Ni} **(μm)**	λ_{pore} **(μm)**	λ_{YSZ} **(μm)**	φ_{Ni}	φ_{YSZ}	φ_{Pores}	L_{voxel} **(μm)**	L_{RVE} **(μm)**	***N***	***R***
R	1	5	0.6	0.4	0.4	.27	.33	.40	0.60	12	20	20
	2	5							0.40			30
	3	5							0.30			40
	4	4							0.24			50
	5	1							0.20			60
N	6	1	0.6	0.4	0.4	.27	.33	.40	0.24	7.2	12	30
	7	5								9.6	16	40
	8	1								14.4	24	60
λ_{pore}	9	3	0.6	0.8	0.4	.27	.33	.40	.27	16	20	60
λ_{YSZ}	10	3	0.6	0.4	0.8	.27	.33	.40	.27	16	20	60
λ_{Ni}	11	2	0.9	0.4	0.4	.27	.33	.40	.27	16	18	60
φ_i	12	1	0.6	0.4	0.4	.35	.43	.22	0.24	12	20	50

6.3. Results

6.3.1. Discretization and RVE size

Previously, it was found that discretization influenced accuracy and that RVE size controlled standard deviation of the modulus (refer to Chapter Four). For nonlinear behaviors, the goal is to capture the trend in convergence behavior in order to be reasonably confident of accurate results. Figure 6.3 plots the mean with outliers to the

maximum and minimum values for modulus in plot (a) and yield stress in plot (b). On the right hand side of the plot is an additional reconstruction with a smaller RVE size. Model numbers five and eight, with $R = 60$ and $N = 20$, are not shown in Figure 6.3, but have yield stresses of 40.33 MPa and 37.78 MPa, respectively.



(a) (b)

Figure 6.3. Variation in modulus (a) and yield stress (b).

Due to the inherently larger variation in stress-strain curves, it is hard to determine from Figure 6.3 the best model for analysis. The need for accuracy must also be balanced with the significantly larger computation times required to analyze a more refined microstructure. In Figure 6.4 and Figure 6.5, contour plots are shown for realization cross-sections with increasing RVE sizes, but with a constant element size. The figures are scaled to show the size relationship between images. The stresses are obviously dependent on the loading in the x-direction. In the smaller sizes in Figure 6.4, higher stresses appear to be more heavily distributed at the edges of the microstructure.

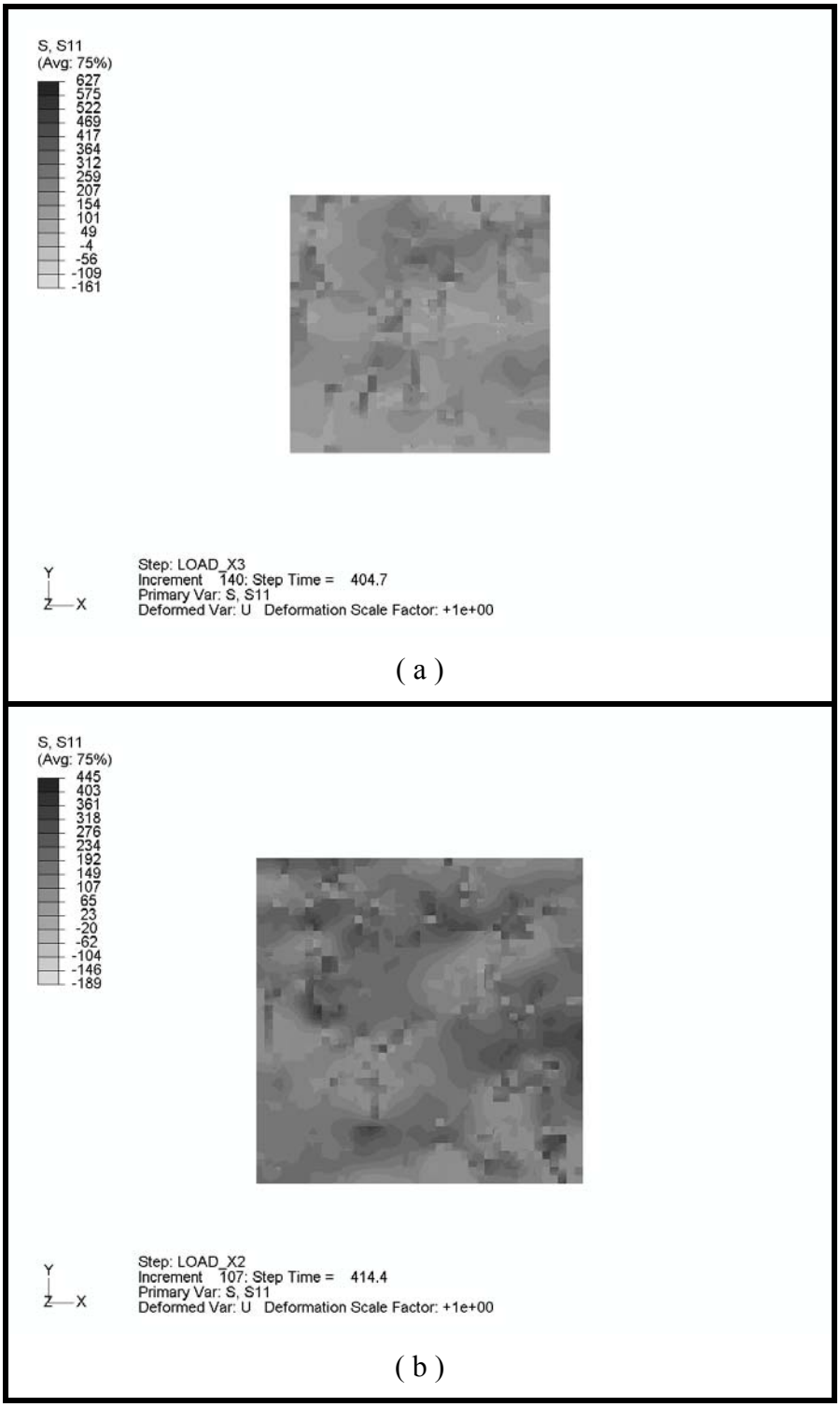


Figure 6.4. Stress contour plots for (a) *R=30; N =12* and (b) *R=40; N =16.*

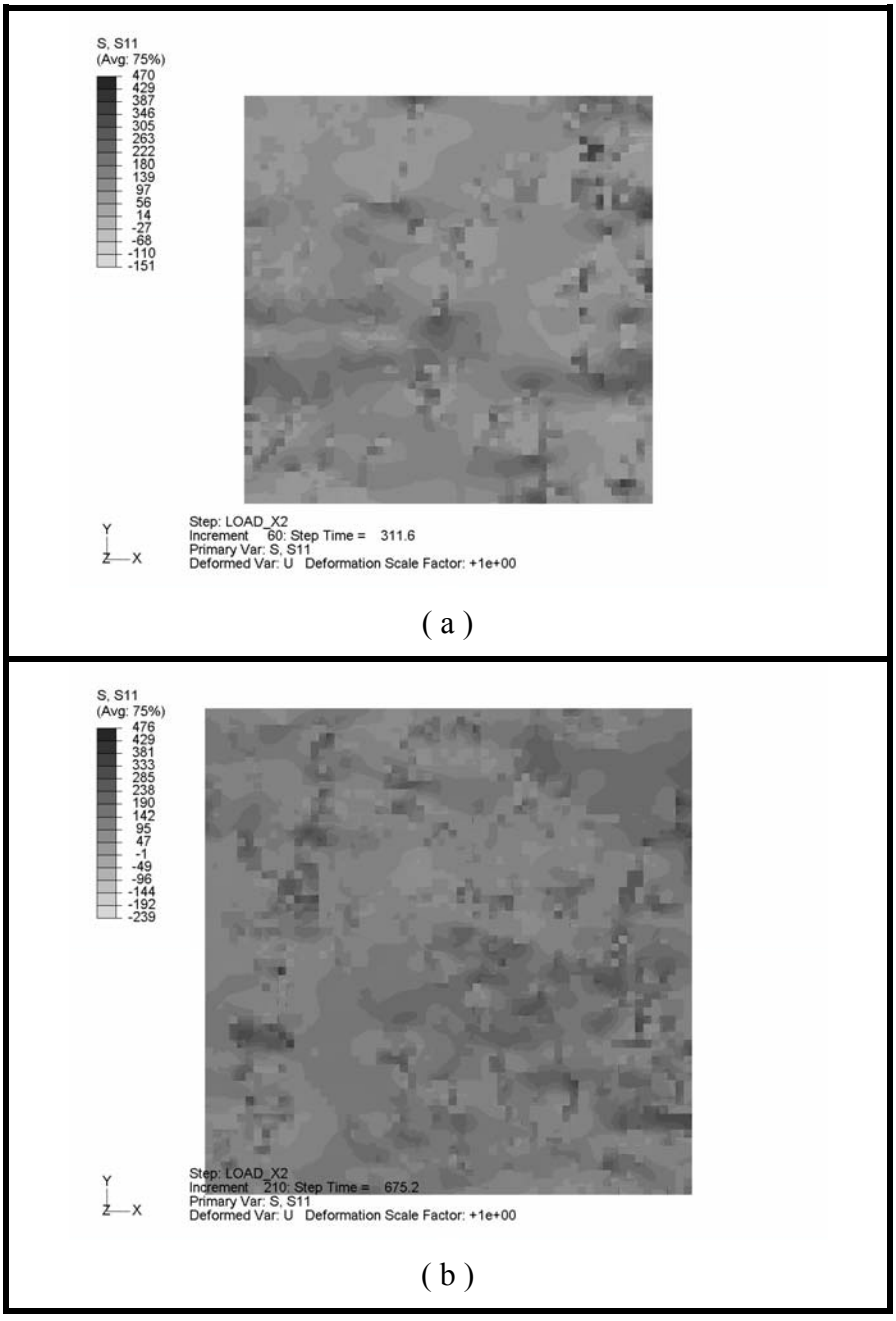


Figure 6.5. Stress contour plots for (a) *R=50; N =20* and (b) *R=60; N =24.*

While the contour plots provide a qualitative clue about stress behavior, the mark functions in Figure 6.6 provide a quantitative measure of this edge effect. For four different RVE sizes, at the point of yield, the mark function is calculated with the binning parameter set for the x-direction stresses to be greater than four times the yield stress. Since it is for all phases, including the pore phase, it is normalized by one and literally provides the percentage for the entire volume.

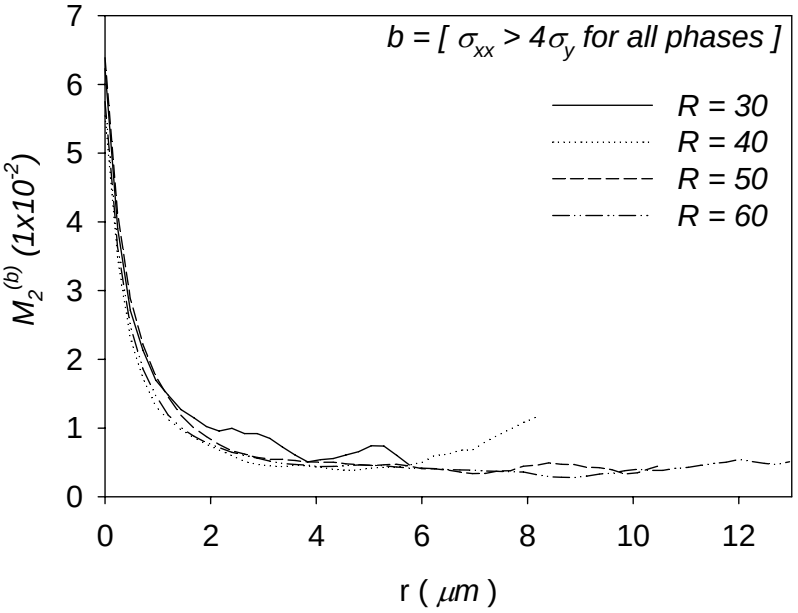


Figure 6.6. Mark probability function for different RVE sizes at yield.

Finally, plots of the stress-strain curves are shown in Figure 6.7 for sets 3-4 in Table 4.1. The plot compares the stress-strain curves of RVEs of the same size but different discretization, where the maximum and minimum curves for each realization set are shown. The middle section of grey is the overlap between the two different RVE sizes. The overlap between the stress-strain curves for the two different sets of realizations shows that the deformation behavior is inherent to a particular microstructure

and independent of RVE size and voxel size. This is so long as RVE size and voxel size are sufficient. Although the smaller element size for the $R = 50$ realizations seems slightly more capable of capturing a higher amount of damage and plastic behavior in the composite, as shown by its lower bound in Figure 6.7. The stress-strain curve of a random distribution with 40% porosity is also included since it provides insight into the impact of microstructure order for the damage and plasticity models. The much higher stress-strain curves for the random microstructure highlight that pore size and the arrangement of the nickel and YSZ do influence stress-strain behavior, much like results for modulus in Figure 4.5.

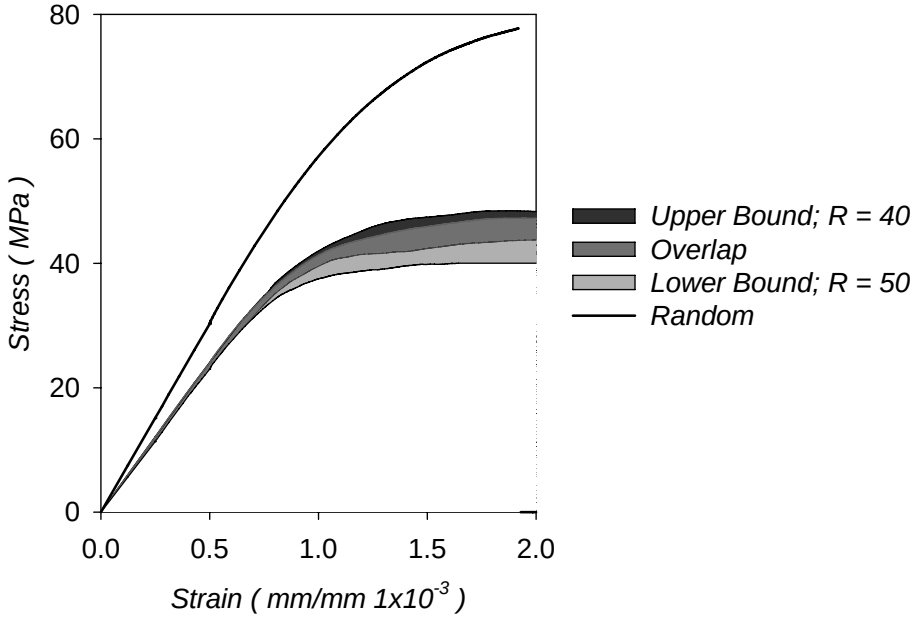


Figure 6.7. Stress-strain curves for different realizations sets of 40% Ni-YSZ.

6.3.2. Microstructure analysis

6.3.2.1. Base model

Once the RVE is determined, the first step is extensive examination of the base model, the 40% porous microstructure with the same features as the original ORNL sample. The model was from set 4 in Table 6.1 and has a yield stress nearest the average yield for the realizations. It was strained until a fully plastic state occurred, and then each of the tools described in sections 6.1and 6.2.3 were applied. First, using equation (6.3), the stress-strain curves for the nickel and YSZ phase are plotted in Figure 6.8. Next, mark functions for the stresses in all three phases (Figure 6.9) and then the mark functions for yield and damage in only the nickel and YSZ phases (Figure 6.10) are studied. Each mark function is shown for the direction of loading (x-direction), and the direction normal to loading (yz-plane), and can be seen to vary depending on direction. In Figure 6.10 the nickel and YSZ probability distributions are also shown for comparison purposes. The plots show the amount and interaction of the field behaviors.

Finally, the stress distributions occurring in nickel and YSZ are shown in Figure 6.11(a-b). In Figure 6.11 (c-d), the stress distributions are plotted for only the damaged or plastic regions in the base model. In Figure 6.11 plots (a) and (d) fit a Weibull distribution, while YSZ in (b) did not fit a normal or Weibull distribution and was left as a histogram. Only the areas of plastic strain for nickel matched a Gaussian distribution. For increasing strains the overall stresses in (a) and (b) changed very little compared to the significant changes seen in the plastic and damage zones in (c) and (d). The shift in curves for (c) and (d) are partly due to redistribution of stresses, but also the increase in the average effective stress, σ_o, within the composite.

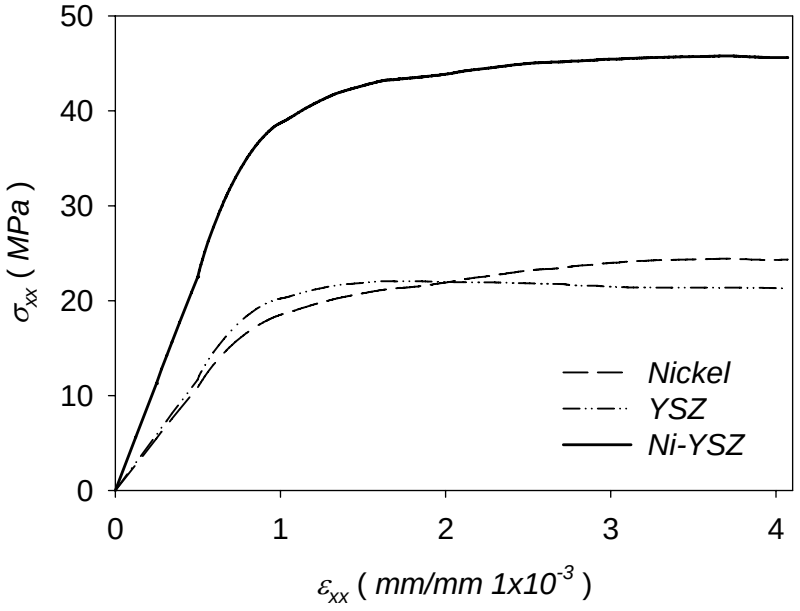


Figure 6.8. Average stresses carried by nickel and YSZ for base model.

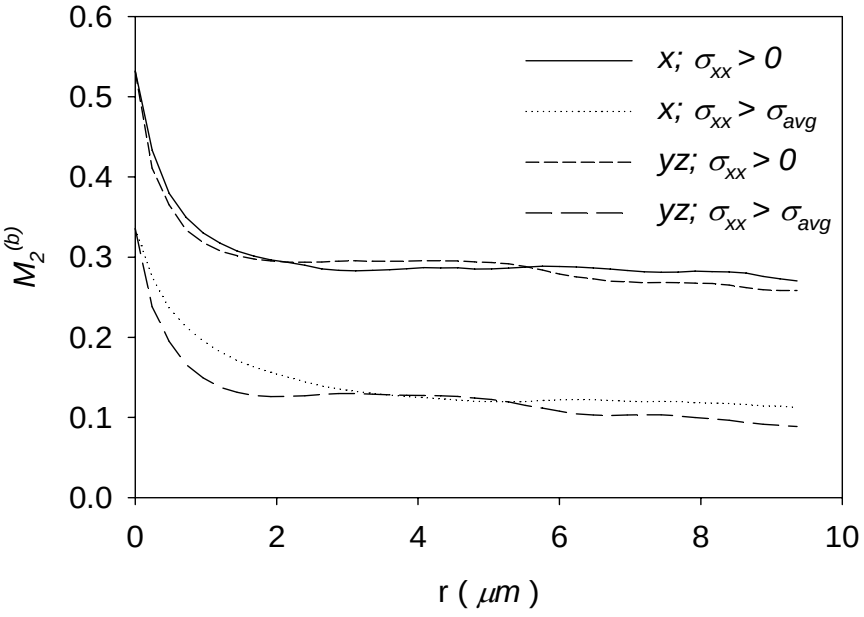


Figure 6.9. Mark function for all phases at point of yield for base model.

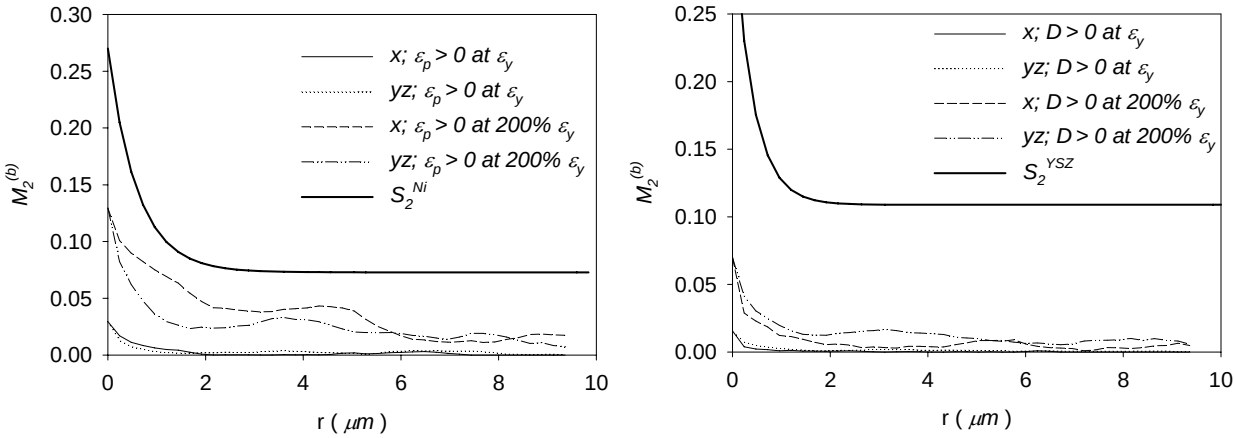


Figure 6.10. Mark functions for (a) plastic strain in Ni and (b) damage in YSZ.

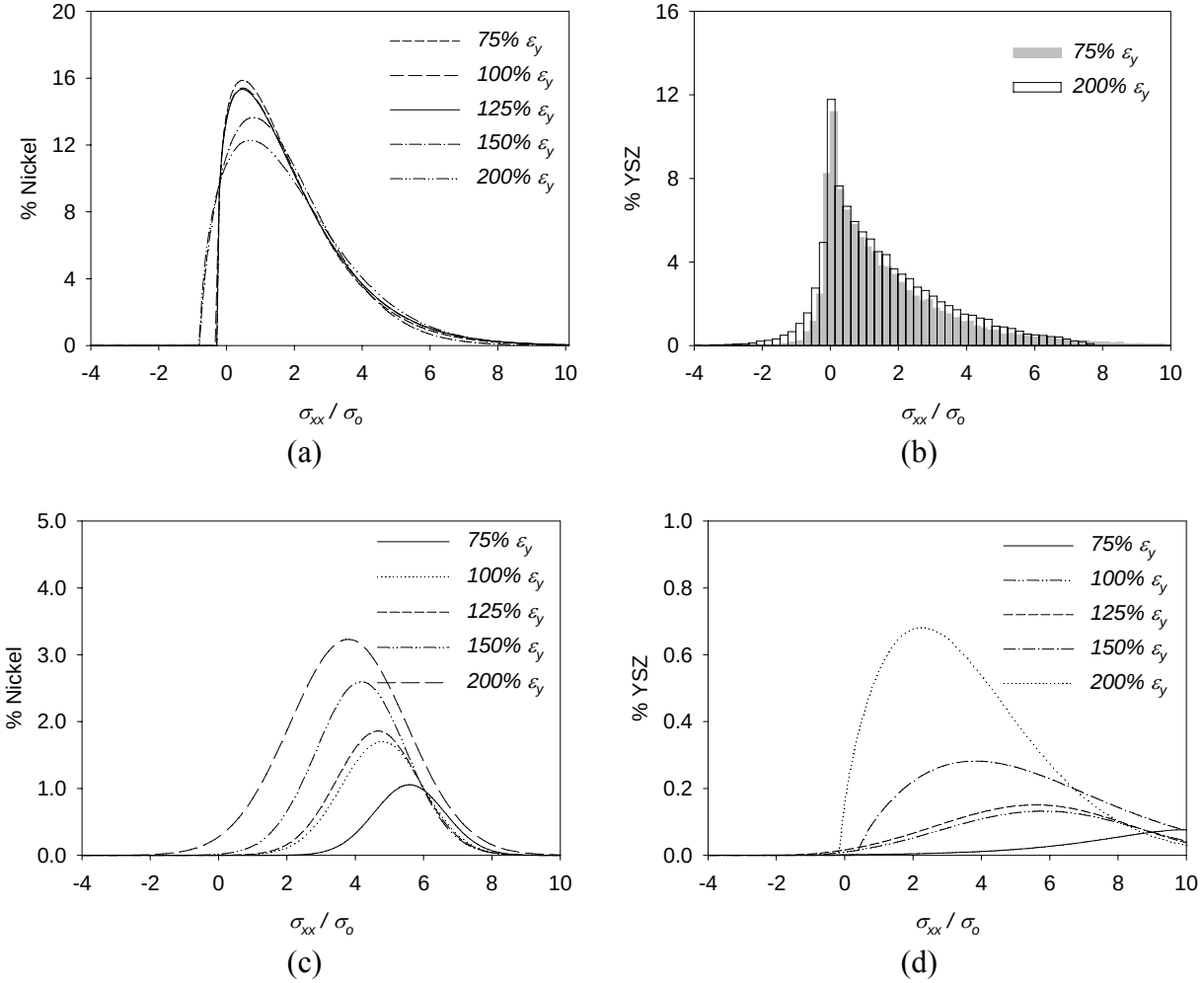


Figure 6.11. σ_{xx} for Ni (a) ,YSZ (b), plastic (c) and damage zones (d) at multiple strains.

6.3.2.2. Porosity

The damage and plasticity behavior for a 22% porous model was also calculated, and the yield stress was found to be 101.47 MPa. When the stress-strain curves for the 22% model and the 40% model are normalized by their respective yield stresses, the curves are very similar, as shown in Figure 6.12.

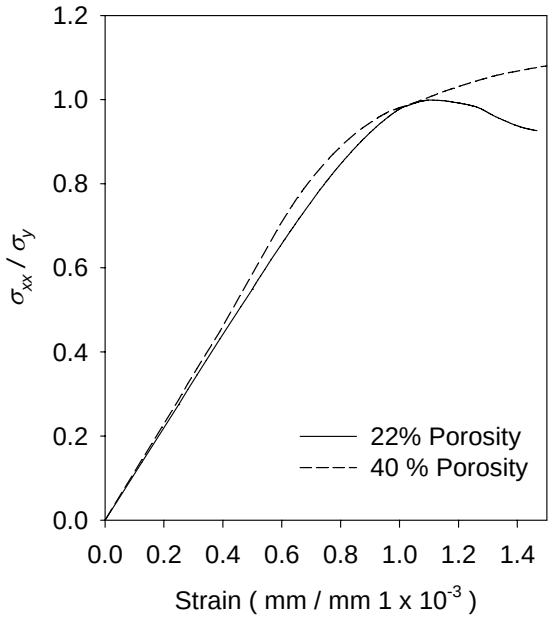


Figure 6.12. Stress-strain curves for 22% and 40% porosity.

6.3.2.3. Microstructural variation

In Table 6.1, four different microstructures are listed with changing characteristic lengths of the Debye random media for each of the phases. Stress-strain curves for these different realizations are plotted against the base model (Figure 6.13). While changes are not large, some variations can be seen, such as the early flattening of the curve when the characteristic length of nickel is increased. There is also an increase in strength with an

increase in YSZ. While the stress-strain curve is slightly higher for the increased pore length scale, it has basically the same shape as the base model.

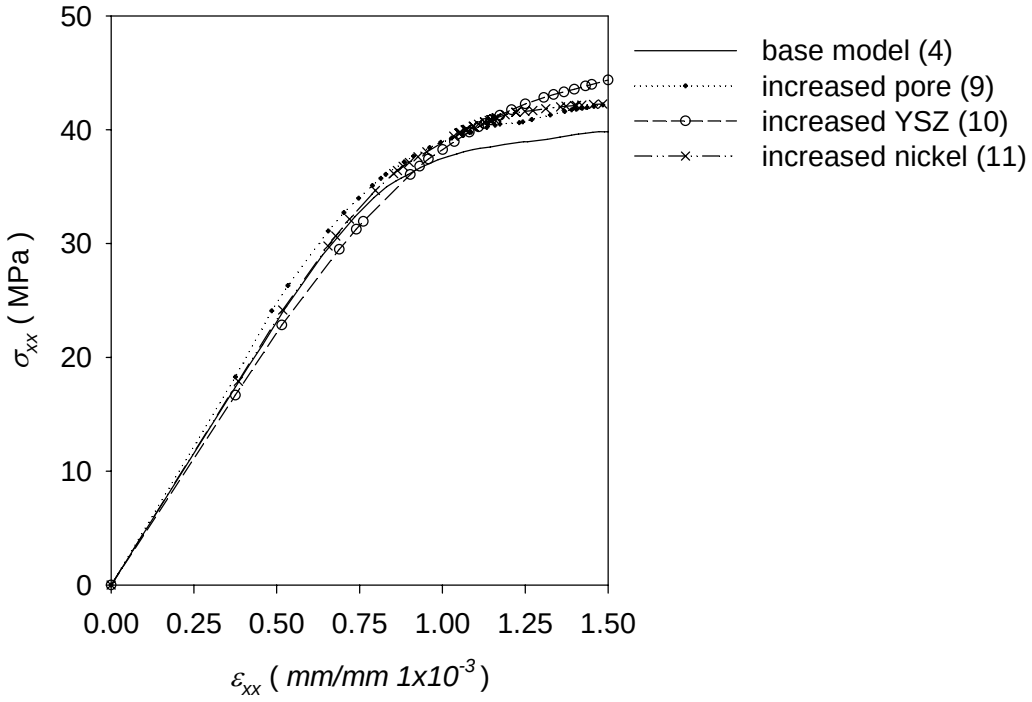
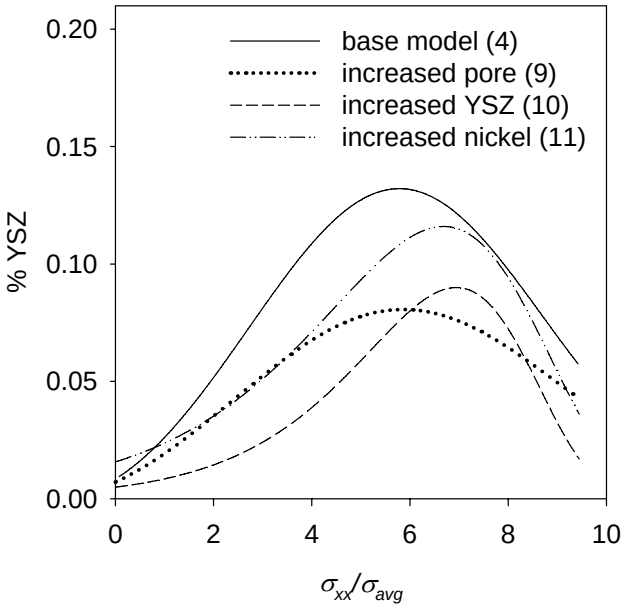
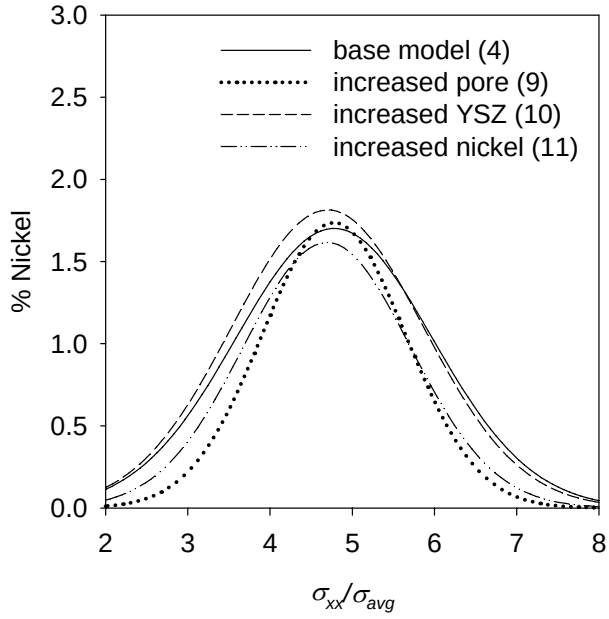


Figure 6.13. Stress-strain curves for changing characteristic lengths.

To gain more insight into the stress-strain curves in Figure 6.13, the distribution plots are found for the percentages of nickel and YSZ experiencing plastic strain and damage, respectively. The plots are similar to those in Figure 6.11(c-d) except only the stress distributions at yield are shown, and multiple realizations are compared. Once again, nickel matched a Gaussian distribution and YSZ was a Weibull curve. In Figure 6.14(a) the shapes of the base model and increased pore curves are the same even though the magnitude differs, and in Figure 6.14(b), the peaks match for the two curves.



(a)



(b)

Figure 6.14. Distribution fits for damaged YSZ(a) and plastic nickel(b) at yielding.

6.4. Discussion

6.4.1. RVE size and discretization

The increased variability in the RVE size and discretization results means that the methodology from Chapter 4 is not necessarily the best way to determine RVE size. It is certainly the least efficient for the larger computer models. In fact, Figure 6.3 shows that the variation of yield stress is much larger than the modulus even for the small sample sizes. However, the figure does show that smaller RVE sizes have a negative impact on both accuracy and standard deviation. Recall that very small RVE sizes accurately predicted linear material properties. Instead, the ability to study stress distributions within the microstructure through histograms and mark functions provides more insight into the microstructure.

In the plots of the mark functions for stresses in all phases (Figure 6.6), the larger sizes of $R = 50$ and $R = 60$ reach a fairly smooth LRO as compared to that of the smaller sizes. This is especially true for $R = 30$, which exhibits an almost periodic nature across its length. When the curves are compared to the equation for Debye random media (eqn. 2.2), the curves match fairly well with a characteristic length of 0.6 μm, especially for the larger sizes. Overall, the mark correlation function appears to show the influence of edge effects, and when the binning stress is high enough, the curve acts similar to a two-point probability function.

6.4.2. Data analysis of base model

Figure 6.8 through Figure 6.11 provide a wealth of information about the internal stresses occurring in the microstructure. They feature stress decomposition, mark

functions, and probability distributions, and each figure provides a different insight into the cermet's behavior. To begin with, the stress decomposition of Figure 6.8 showed that nickel and YSZ carry very similar loads, but at a certain point, YSZ does not carry an increasing load. Contrast this with the fact that only 7.0% of YSZ, equivalent to 2.3% of the entire sample volume, even experiences damage at two times the strain at yield. On the other hand, 12.9% of nickel undergoes a slightly higher occurrence of plasticity. However, due to the smaller volume fraction of nickel, this represents only 3.5% of the entire volume.

The distribution of damage and plasticity illustrates the non-uniform stress distribution within the microstructure, and at yield, thirteen percent of the solid mass is carrying a negative or zero load. The mark function in Figure 6.9 verifies that little more than fifty percent of the total volume is carrying a tensile load, and this tensile load is fairly independent of the loading direction, the x-directions, and the cross-planes, the y and z-directions. Therefore, this part of the microstructure could be considered "fully connected," such that this region will always carry an initial load regardless of size. However, when the binning value for stress is above the average stress, the mark function obviously changes with direction, although both curves have the same LRO behavior. When fitted to the Debye equation, the characteristic length of the curve in the x-direction is twice that of the cross-plane direction. This means that the higher stresses are more closely grouped and continuous in the direction of loading. As the stress spreads away from areas with high stress, the grouping becomes smaller and shorter in range. It is also noteworthy that the characteristic length in the cross-planes and for the tensile stress curves is between 0.55 – 0.60 μm, a value in between the length scales of the YSZ

and nickel phases. This directly links the stress distributions occurring to the specific microstructure.

In Figure 6.10, the mark functions for the occurrence of damage and plasticity provide information about the occurrence of specific failures. At the yield strain, the relationship between sites of plasticity and damage is vanishingly small at 2 μm. At two times the yield strain, the plasticity of nickel shows no particular order in either the short or long range. The curves in the yz-plane are still less than those in the x-directions, which corresponds to the stress distributions discussed previously. Damage is much more localized than plasticity in all directions. Interestingly, it also appears damage is slightly higher in the cross-plane than in the direction of loading. Most likely, damage occurs at a point and then spreads perpendicular to the initial damage site, which corresponds with the maximum stress criteria used for YSZ.

The last data tool was distribution curve fitting of histograms, and Figure 6.11 used four different distributions to study the microstructure. Both nickel and YSZ have large right leaning stress distributions (Figure 6.11 (a) and (b)), which result from the lack of compressive stresses in the microstructure. The fact that YSZ does not fit a Weibull distribution results from the bulk of the stress being near the average stress value in the microstructure. This is probably due to its larger volume fraction providing a slightly more uniform stress distribution. Surprisingly, neither the nickel nor YSZ distribution change significantly in shape or size with increasing strain, meaning that even as plasticity and damage occur the overall stress interactions between the phases stay the same. The strong right leaning distributions correspond well to the notion of failure in materials. Defects lead to areas of high stresses, which then lead to failure, and

although defects are not specifically modeled the distribution of Ni, YSZ, and pores lead to similar behavior. It would also be expected that for different microstructures different stress distributions would result. For instance, a completely random distribution would probably have a Gaussian distribution since each phase is randomly distributed.

In contrast, in Figure 6.11(c) and (d), the distributions for the plastic and damage zones change drastically, with increasing strains, mainly in size. Nickel has a Gaussian distribution for all strains and the shifting of the curve to the left for increasing strain comes from the change in stress distribution from previously deformed elements. Looking at Figure 6.11 (a) for increasing strains, the Gaussian distribution slowly starts to show a higher left skew. This comes from high stress being "disconnected" from the model as damage and plasticity increase. The YSZ distributions of Figure 6.11 behave exactly as expected, since as each element continues to undergo damage, the stresses continually decrease.

6.4.3. Microstructure variation

The microstructure was varied either by volume fraction or internal length scales. Changing the volume fraction for the microstructure showed that the calculated yield stress was higher, but that the actual shape of the stress-strain curve was similar at least up to yielding. The 22% porous realization is the only simulation to actually show failure at any point, which for the kinematic load, occurred near the edge and spread in the yz-plane. This makes sense as damage growth occurs in the cross-plane. An examination of the mark curve (plot not shown) for binning with all stresses greater than the average current stress found that the SRO matched that of a Debye function with a 0.6 μm. However, the LRO order was greater than that of the Debye function, suggesting that the

stresses are correlated at a much larger length scale for the smaller porosity microstructure.

When changing the internal length scales, the goal was to change the actual shape of the stress-strain curves in some significant way, and from Figure 6.13, the most significant change occurs with a change in YSZ length scales. The curves with increased YSZ exhibited better post yielding behavior. Looking at Figure 6.14(a), it also has the smallest and most right leaning distribution. Apparently the grouping of YSZ in larger clusters prohibits damage growth. Yielding in nickel, as shown in Figure 6.14(b), is fairly insensitive to changes in length scales, although the largest curve is the increased YSZ model. Therefore, larger YSZ particles also help shift loading to the more ductile nickel, improving the post yielding behavior. It is of interest that changing length scales had at most a potential 5% change in modulus, but those changes could then result in significant changes in the stress-strain curves

6.5. Summary

The voxel reconstructions of previous chapters were applied to a study of nonlinear deformation. By setting the YSZ phase properties to those for brittle failure and the nickel phase to that of plastic behavior, a yield stress value of 40 MPa was predicted for 40% Ni-YSZ. The analysis found that for RVE size, yield stress was more variable than modulus and a larger RVE size, but not necessarily a smaller voxel size, was needed than was usually the case for a linear analysis.

Then while varying the microstructure and studying in detail one base model, the following conclusions were made about failure in the actual cermet:

- the mark function shows a correlation between microstructure and stress distributions,
- the stresses in each phase have a Weibull type distribution,
- the ratio of yield stress to modulus is similar for 22% and 40% porosities, and
- a slight change in YSZ length scales has a significant impact on the stress-strain curves.

Future work on this area would focus on deformation behavior after yielding. While examined in this chapter, much remains unknown about how the constituents, especially nickel, would actually change after initial yielding or damage. Implementation of an incremental plasticity theory would better model unloading behavior and address the fact that the stress distribution within nickel is only relatively monotonic. Another aspect of interest would be examination of fracture in the cermet, although published experimental data exists for fracture properties of the anode [28, 33, 87]. Experimental information on the stress-strain behavior of the anode would be useful, though Radovic and Lara-Curzio did find bi-axial strength values 63.8 ± 19.7 MPa [31]. While this does not correspond exactly with yield stress behavior, the values predicted within this work appear reasonable.

CHAPTER 7

TIME-DEPENDENT DEFORMATION

As discussed in the Introduction (recall Figure 1.2) stresses in the fuel cell are dependent on a complex interaction of manufacturing, operating conditions, and cell configuration. However, even without external loading the thermal mismatch of the electrolyte and anode bi-layers will still result in a constant stress in the anode material. Then at the most basic level, the cermet will carry internal stresses due to the thermal mismatch of nickel and YSZ. These two conditions, along with the fact that pSOFCs undergo high temperatures for extended periods of time, make the study of time-dependent deformation of nickel within the composite a priority.

There are many unknowns concerning the time-dependent, or creep, deformation of the anode material. Both numerical and experimental studies of the anode-electrolyte bi-layers have found that the residual stresses in the electrolyte are in compression resulting from the initial sintering at high temperatures [34, 103]. Lara-Curzio et al. found an initial drop in these residual stresses at 800°C due to creep deformation of Ni-YSZ [34]. Gutierrez-Mora et al. specifically studied creep in the bi-layer and found that at high temperatures (> 1100ºC) nickel controlled deformation, but that the bi-layer deformation did not correlate with nickel stress exponents [35]. Other studies of nickel aggregation primarily focused on electrical performance and not structural properties [18, 21, 23].

The voxel reconstructions were used to examine the cermet properties independent of the bonded electrolyte layer. By examining an initially stress-free anode with different applied strains and strain rates conclusions can be extrapolated to possible PEN behavior. This chapter will first introduce the nature of stresses in the PEN layer that result from thermal mismatch during the sintering process. Then the stresses from thermal mismatch in the anode itself will be discussed. Next, the methodology behind creep in pure metals and composites will be covered before determination of RVE size. An abbreviated study of RVE size for creep deformation is done, and the different measurements to determine convergence of the stress-strain curves are discussed.

Once RVE size is determined, four different features of creep in the anode material are investigated: stress relaxation over time, strain rate dependence, the significance of YSZ percolation, and finally the influence of the internal nickel length scales. Each of these topics provides insight into the final deformation experienced by the PEN layer.

7.1. Theory

7.1.1. Thermal stresses

The anode-electrolyte bi-layer is sintered at temperatures as high as 1400°C and as the bi-layer is cooled to room temperature, the lower CTE of YSZ, α_{YSZ}, results in a compressive stress in YSZ and a tensile stress in the anode [19]. Figure 7.1 shows the resulting warpage that occurs during the cooling process, where higher shrinkage in nickel deforms the electrolyte. Since sintering occurs at temperatures higher than operating temperatures, this stress distribution will occur at all operating temperatures.

Experiments have found these electrolyte compressive stresses range from 500-800MPa even at high temperatures [34, 103]. Recall that the electrolyte typically has heights of 10μm compared to 1mm for the anode, meaning the anode will carry significantly less stress. For the previous dimensions and for a compressive electrolyte stress of 800MPa, the stress in the anode would be 8MPa.

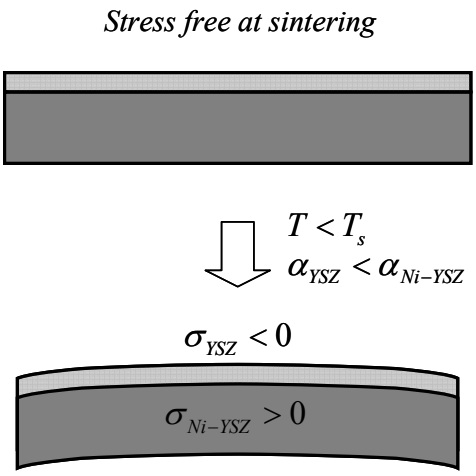


Figure 7.1. Residual stresses in the bi-layer.

If the anode is heated without an external load, the distribution of stresses within the composite will vary around an average result. In fact, since the CTE of nickel is greater than that of YSZ, the average stresses in YSZ will be positive compared to negative stresses in nickel. Figure 7.2 illustrates the difference in average stresses in each phase for a temperature increase without additional loading. In Figure 7.2 both nickel and YSZ are purely elastic and the model is idealized in that PEN residual stresses are not incorporated. The magnitude of the average stresses is dictated by the volume fractions of each phase such that equilibrium is satisfied. Also at the Curie point, the relationship to the composite CTE can be seen with the mirroring of the stress spikes due to nickel's

paramagnetic transition. The creep deformation of nickel will therefore vary throughout the microstructures due to this distribution of stresses. As nickel deforms so will the equilibrium relationship between nickel and YSZ. The question then becomes what is the overall impact of nickel deformation on the anode.

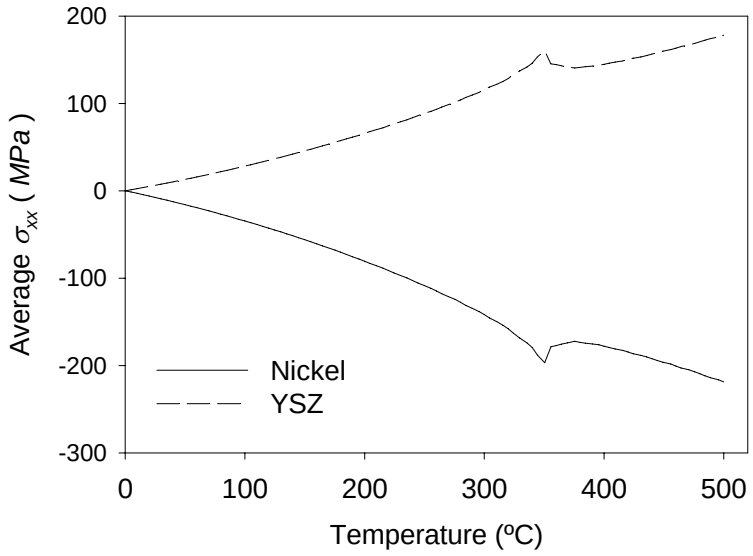


Figure 7.2. Internal stresses in Ni-YSZ for stress free temperature increase.

7.1.2. Composite creep

Since in comparison to nickel, YSZ will experience little deformation over time, it can be assumed that temperature-dependent deformation of Ni-YSZ will be controlled by the creep behavior of the nickel phase. First though, we present a temperature-dependent definition of strain for the composite as shown in equation (7.1). This definition is for the bulk response of the composite, even though, as illustrated in Figure 7.2, internal stresses will exist even at a stress-free stage for the composite. For small times the total strain of

the composite, ε^{tot}, is the sum of the elastic, ε, and thermal strains, ε^{thm}, occurring in the composite such that

$$\varepsilon^{tot} = \varepsilon + \varepsilon^{thm} . \tag{7.1}$$

Therefore Hooke's law for a uniaxial loading must account for the thermal deformation, and total strain is now defined as

$$\varepsilon^{tot} = \frac{\sigma}{E} + \alpha \Delta T . \tag{7.2}$$

Over extended time periods, the nickel phase will begin to experience creep and the total strain must now include creep strain, ε^{c}, shown in (7.3),

$$\varepsilon^{tot} = \varepsilon + \varepsilon^{thm} + \varepsilon^{c} . \tag{7.3}$$

In metals, creep can typically be broken down into three stages, an initial fast stage of strain, a long term steady-state deformation, and finally, a fast deformation preceeding catastrophic failure, as illustrated in Figure 7.3 (a). The classic approach to creep in metals uses an Arrhenius equation to describe the steady-state strain rate, which is defined in (7.4).

$$\dot{\varepsilon} = A\sigma^{n} e^{-Q/RT} \text{, where} \tag{7.4}$$

A and n are dimensionless constants. Q is the activation energy with units of calorie per mole. R is the universal gas constant and T is the temperature. The strain rate is the derivative of strain with respect to time of the total strain, defined as

$$\dot{\varepsilon} = \frac{d\varepsilon^{c}}{dt} . \tag{7.5}$$

Since both thermal and elastic strains are independent of time, they drop from the equation(7.5). In studying creep, the n exponent is often of primary interest since it describes the slope of the log strain rate versus stress curve (see Figure 7.3 (b)).

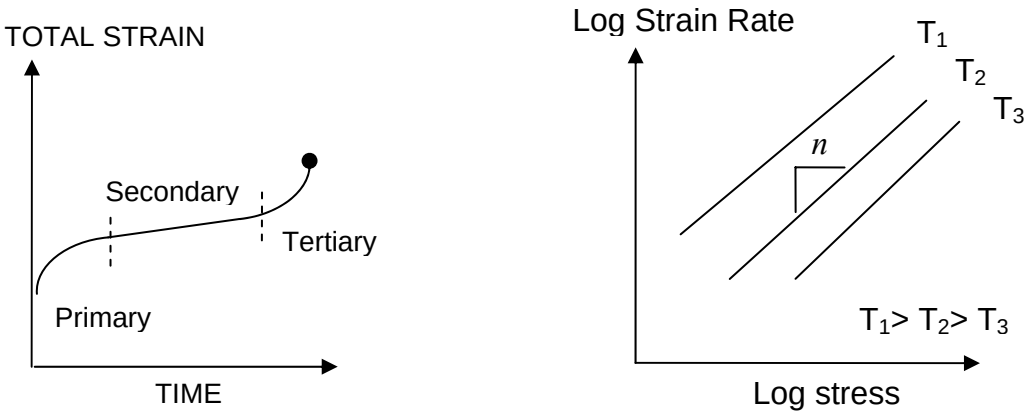


Figure 7.3. Illustration of stages of creep (a) and log stress-strain curves (b).

For a pure material, once steady-state creep has been reached, the stress will become constant with time. This is not necessarily true for a metal-ceramic composite. Initially a stress-strain curve of the composite will have an instantaneous slope equal to the modulus of the microstructure, E_i, but over time this slope will change. Assuming that creep in the ceramic phase is minimal, once the steady state strain is reached in the time-dependent phase, the microstructure will continue to deform by the rules governing the remaining phase. In Figure 7.4, the stress-strain curves from an FE analysis of pure nickel and a Ni-YSZ composite are plotted for a constant strain rate and temperature. It can be seen that over time as nickel experiences creep, a steady-state modulus, E_{ss}, value is reached for the composite. This value is equivalent to the modulus of the YSZ portion of the composite. Since the load is applied with a constant strain rate, the curve would have the same behavior when plotted against time.

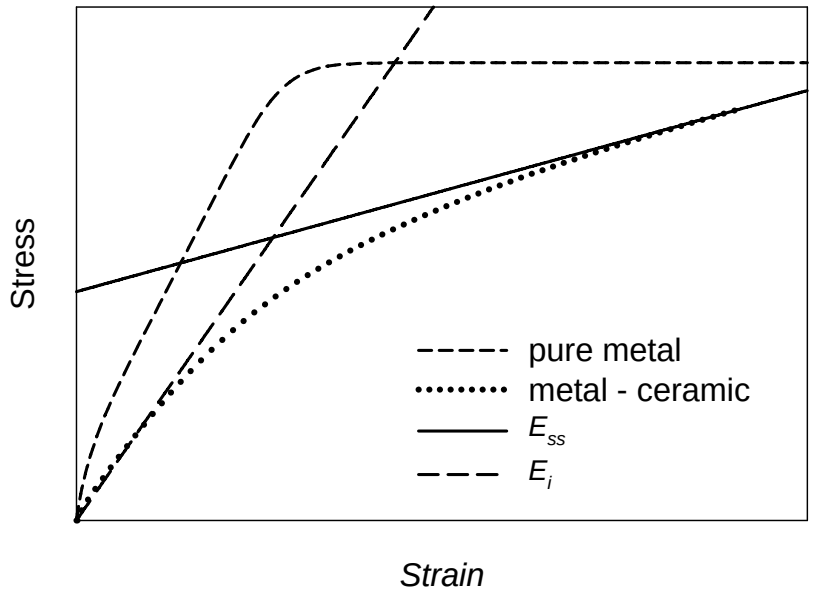


Figure 7.4. Illustration of composite creep for a constant strain rate and temperature.

7.2. FE model

In the analysis of time-dependent deformation, several models of different element and RVE sizes are examined first, followed with reconstructions that vary either porosity or internal length scales. A complete list of the reconstructed models is provided in Table 7.1.

The YSZ material is the same elastic material from Chapter Four, but the nickel material is modified to account for time-dependent deformation. A temperature-dependent modulus and CTE is used for nickel, while a steady-state creep formulation is used based on equation (7.4). The parameters for nickel creep were determined from multiple published works [104-108]. Weertman and Shahinian found the creep exponent, n, to be 4.6, which was used for nickel in this work [105]. The material data is also

modified to account for a temperature-dependent modulus. The complete property data is found in Appendix D. All simulations are run at 500°C unless otherwise specified.

Two different loading conditions are used throughout. The stress relaxation model loads the model to an initial strain in the x-direction and a constant temperature for the entire model. The model is then held at a constant strain and temperature over time to allow stress relaxation to occur. The second model applies an increasing strain over time in the x-direction for a constant temperature. The strain is applied so that the strain rate for the composite is constant over time.

Table 7.1. Time-dependent microstructure realizations.

Mod.	**Set**	λ_{Ni} **(μm)**	λ_{pore} **(μm)**	λ_{YSZ} **(μm)**	φ_{Ni}	φ_{YSZ}	φ_{pores}	L_{voxel} **(μm)**	L_{RVE} **(μm)**	***N***	***R***
R	1	0.6	0.4	0.4	.27	.33	.40	0.40	12	20	30
	2							0.30			40
	3							0.24			50
N	4	0.6	0.4	0.4	.27	.33	.40	0.24	7.2	12	30
	5							0.24	9.6	16	40
φ_{YSZ}	6	0.6	0.4	0.4	.46	.14	.40	.30	12	20	40
	7				.44	.16	.40				
	8				.42	.18	.40				
	9				.36	.24	.40				
	10				.30	.30	.40				
λ_{Ni}	11	0.9	0.4	0.4	.27	.33	.40	0.24	16	18	60
	12	0.3	0.4	0.4	.27	.33	.40	0.20	8	20	40

7.3. Results and discussion

7.3.1. RVE size and discretization

Several assumptions started the analysis of acceptable RVE size and discretization. The first was that since for this analysis, nickel is the only phase undergoing nonlinear deformation, then convergence behavior must be better than that for the damage and plasticity analysis, where both phases experienced nonlinear deformation. The second assumption was that our primary concern is the final steady state deformation of the microstructure, E_{ss}, as illustrated in Figure 7.4. To that end, the same base microstructure from Chapter Six is studied for several different RVE sizes and discretizations. Table 7.1 lists values for the instantaneous modulus, the stress and slope of curves at an arbitrary strain, and finally, steady-state modulus as the strain approaches infinity. Each model is loaded in the x-direction with a constant strain rate. It should be noted that an initial thermal strain was preset for a CTE of 12.07 x 10^{-6} / ºC at 500ºC for all realizations.

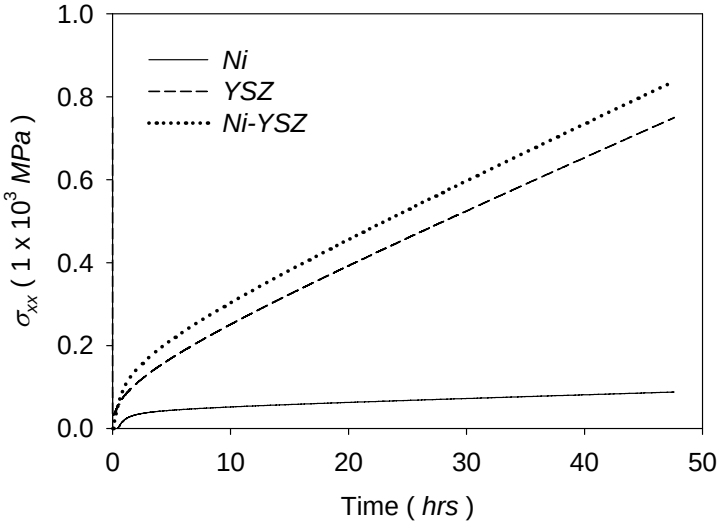
Table 7.2. RVE size and discretization for a strain rate of $1x10^{-6}$ /s at 500ºC.

R	N	#	E_i (GPa)	σ at $\varepsilon = 6.7\times10^{-3}$ (MPa)	E_{ss} at $\varepsilon = 6.7\times10^{-3}$ (GPa)	E_{ss} at $\varepsilon \to \infty$ (GPa)
30	20	1	43.87	148.8	9.29	1.45
		2	42.34	139.8	8.77	1.56
		3	46.16	154.4	9.60	1.71
Average			**44.12 ± 1.92**	**147.7 ± 7.37**	**9.22 ± 0.42**	**1.57 ± 0.13**
40	20	1	45.27	155.9	9.60	2.39
		2	42.07	146.3	9.49	2.49
		3	43.28	150.4	10.25	2.79
Average			**43.54 ± 1.62**	**150.9 ± 4.82**	**9.78 ± 0.41**	**2.56 ± 0.21**
50	20	1	41.77	135.3	7.92	1.43
		2	43.59	144.2	10.8	1.67
		3	41.47	145.6	10.3	2.20
Average			**42.28 ± 1.15**	**141.7 ± 5.59**	**9.67 ± 1.54**	**1.76 ± 0.40**
30	12	1	40.34	137.9	8.68	2.38
		2	41.01	133.4	7.82	1.38
		3	39.88	139.5	8.48	2.93
Average			**40.41 ± 0.57**	**136.9 ± 3.16**	**8.33 ± 0.45**	**2.23 ± 0.79**
40	16	1	41.53	137.6	10.16	2.99
		2	44.71	142.8	9.96	3.54
		3	45.29	138.2	9.30	2.20
Average			**43.84 ± 2.02**	**139.5 ± 2.84**	**9.81 ± 0.45**	**2.91 ± 0.67**

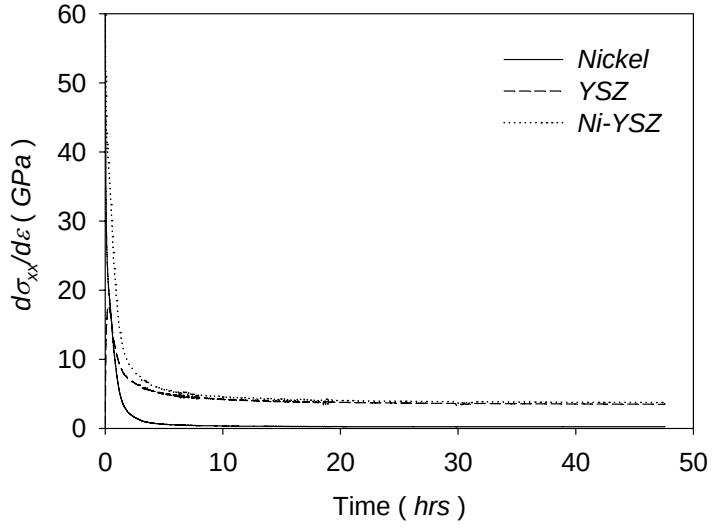
In the data from Table 7.2, one accurate estimate of convergence of the creep, RVE size, is not immediately obvious. As in Chapter Four, the modulus of the microstructure is easily determined for a minimum R and N of 40 and 16, respectively. However, for the stress at a given strain, the RVE size becomes much more significant and the convergence behavior is similar to what was found for the damage and plasticity analysis in the previous chapter. One factor of interest is that the early steady-state

modulus values show a more reliable convergence behavior than that of the infinite value. The variability probably results from the increase of length scale for the non-YSZ phases, as nickel no longer has a non-uniform stress distribution. In other words the microstructure becomes similar to a two phase composite of pores and YSZ.

To further investigate this behavior, Figure 7.5 (a) plots the average stresses for both nickel and YSZ for model 1 in Table 7.1. The curve covers a total of 50 hours with a steadily increasing strain at a constant strain rate. It should be noted that this is not a realistic portrayal of cermet behavior, because catastrophic failure would occur at a much lower strain. However, the curve does illustrate that after a reasonably short time, the slopes of the Ni-YSZ and YSZ curves are almost identical. Figure 7.5(b) plots the derivative of the stress with respect to strain, a value with units equivalent to that of Young's modulus. In Figure 7.5(b), the curve approaches zero very quickly, at little more than five hours. To that end, the steady-state slope at reasonably small times in combination with the stress value for a given strain rate appears to be a better measure of convergence behavior, than either the initial modulus or the steady-state value as time approaches infinity.



(a)



(b)

Figure 7.5. Stress decomposition for cermet (a) and the derivative of stress change with respect to strain (b) over time for a strain rate of 1 x 10^{-6} /s at 500°C.

7.3.2. Base model

7.3.2.1. Stress relaxation

From the previous convergence analysis, the highlighted realization from Table 7.2 was used to study the variation in behavior for stress relaxation and strain rate. The first set of FE models loaded the microstructure to the approximate yield stress estimated in Chapter Six and measured the stress relaxation over time. This was done at both 500ºC and 700ºC for a total of 10,000 hours (refer to Figure 7.6). The increase in temperature has a significant impact on the amount of time required to reach steady state and in the initial stress drop. Next, in Figure 7.7, the impact of the initial stress on the stress relaxation is shown at 700ºC. Also of interest is the rise in stress for a pre-stress value of 10 MPa. This is due to nickel being primarily in compression at such low stress values.

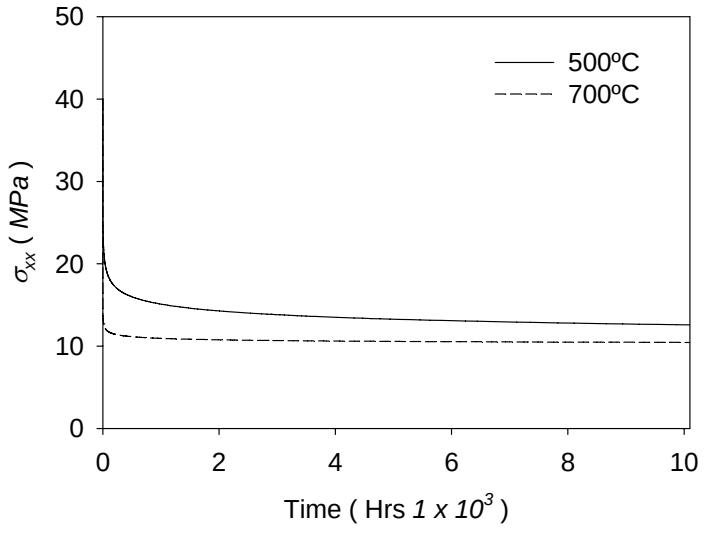


Figure 7.6. Stress relaxation over time for an initial stress of 40MPa.

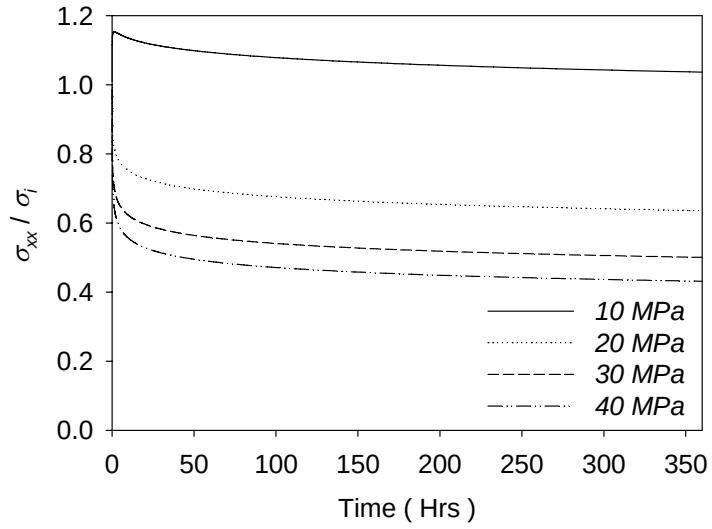


Figure 7.7. Stress relaxation for multiple pre-stresses at 700ºC.

Study of the base model (recall the highlighted realization in Table 7.2) led to several conclusions about the stress behavior of the microstructure. In Figure 7.6 the cermet lost almost seventy-five percent of its initial load in a relatively short amount of time, but the final difference between the two temperatures was only 2MPa. Curve fits to the histograms of stresses at 700ºC, Figure 7.8, support this by showing how the nickel stresses converge at a mean value over time. Also significant is that as nickel experiences creep, the stresses in YSZ shift from a Weibull distribution in Figure 7.8(b) to a Gaussian distribution at 1,000 hours in Figure 7.8(d). This results from the nickel phase no longer carrying the compressive load from thermal expansion. The loading of nickel is also higher than that of YSZ initially, because of its smaller volume fraction. The amount of nickel carrying an initial compression also matters, since, as shown in Figure 7.7, for the smallest load, an initial stress increase occurs.

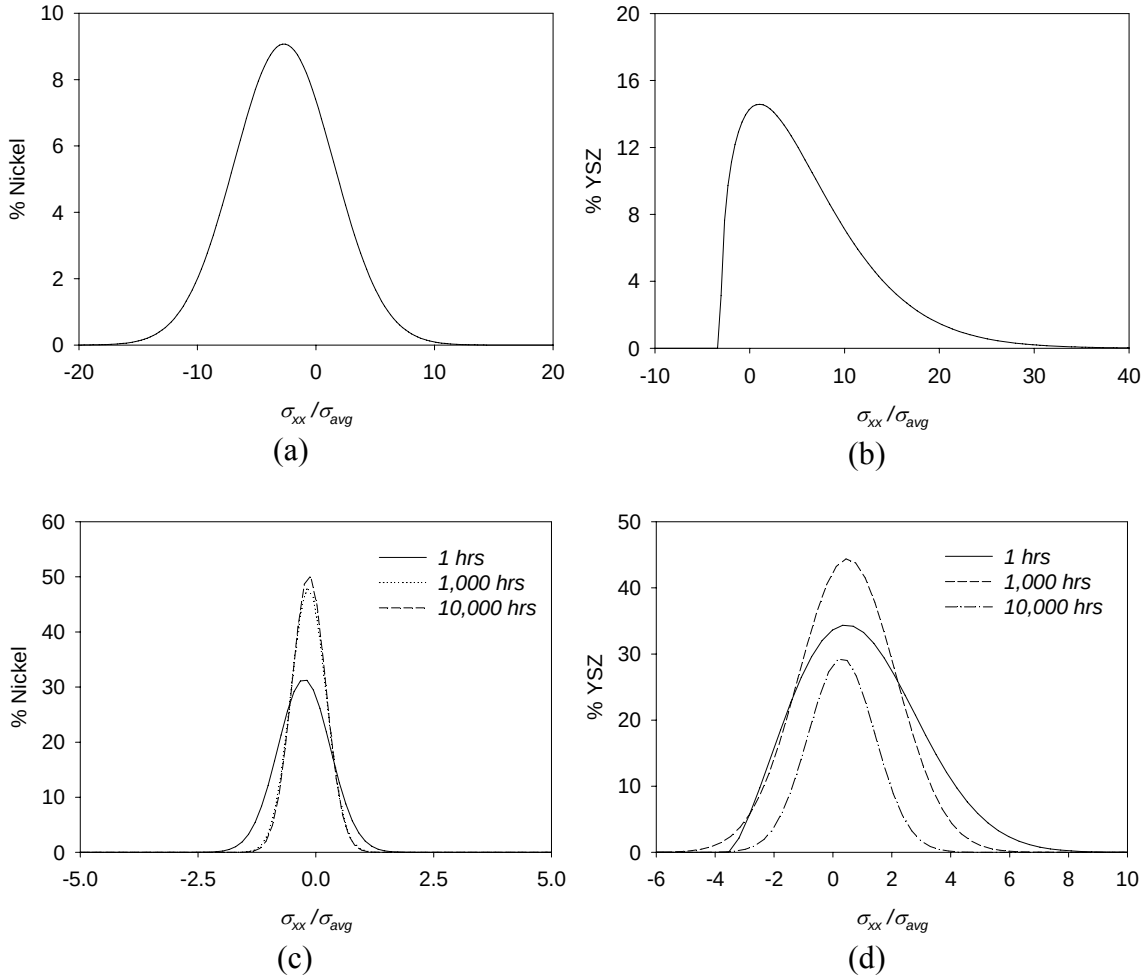


Figure 7.8. Stress distributions for nickel (a) and YSZ (b) at zero time and for increasing times for nickel (c) and YSZ (c) for an initial stress of 40MPa at 700°C.

7.3.2.2. Constant strain rate

Next, traditional stress-strain curves (thermal strain is present, but not plotted) are shown for several different strain rates in Figure 7.9. The stress-strain curves are plotted for eight different strain rates. For the largest strain rate, 1×10^{-1} /s, the total run time is 6.7×10^{-2} seconds and for the smallest strain the total time is 67,000 seconds. As the total run time increases the final stress values decrease as more creep occurs in nickel.

One factor of interest is that all the curves either overlap or intersect at 37 MPa and a strain of 9 x 10^{-4} mm/mm.

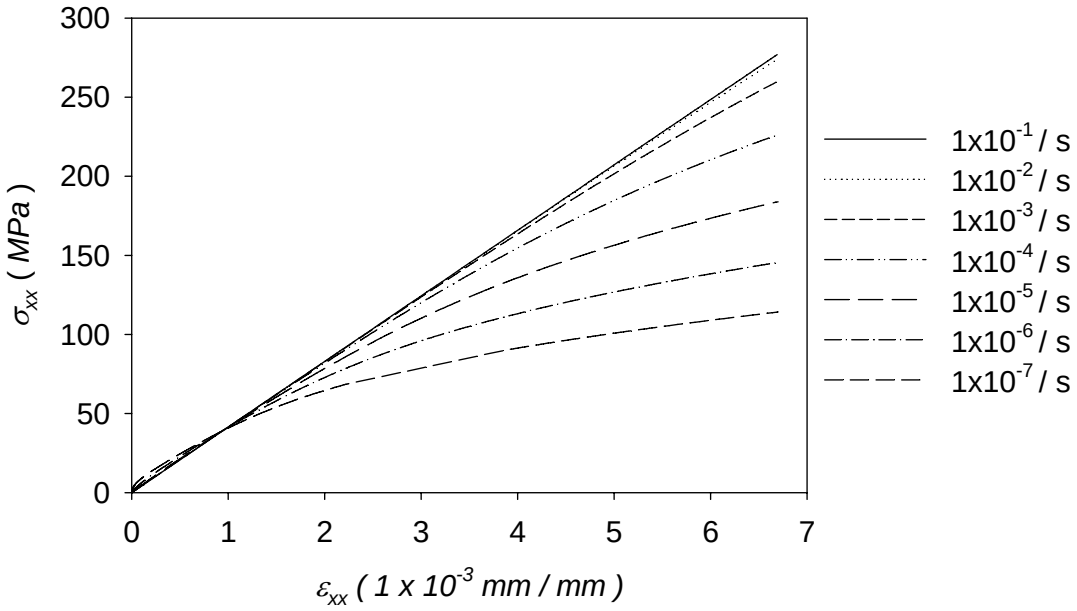


Figure 7.9. Stress-strain curves for different strain rates at 500ºC.

7.3.3. YSZ Percolation

In Chapter Three percolation and the percolation threshold were introduced. Percolation described the connectivity of a phase in the system and percolation threshold was the volume fraction at which a phase could be said to have an infinite cluster. Since YSZ has the same characteristic length as the pore phase, then YSZ should exhibit similar clustering behavior as the pore phase. To that end, the stress-strain curves were examined for a constant porosity of 40%, but a steadily decreasing volume fraction of YSZ. The curves in Figure 7.10 occur for a constant strain rate and temperature. Since CTE will vary significantly with the increase in nickel volume fraction, each realization

was loaded to an initial zero stress for its particular microstructure. It can be seen that as the volume fraction of YSZ drops, the creep behavior of nickel becomes steadily more dominant.

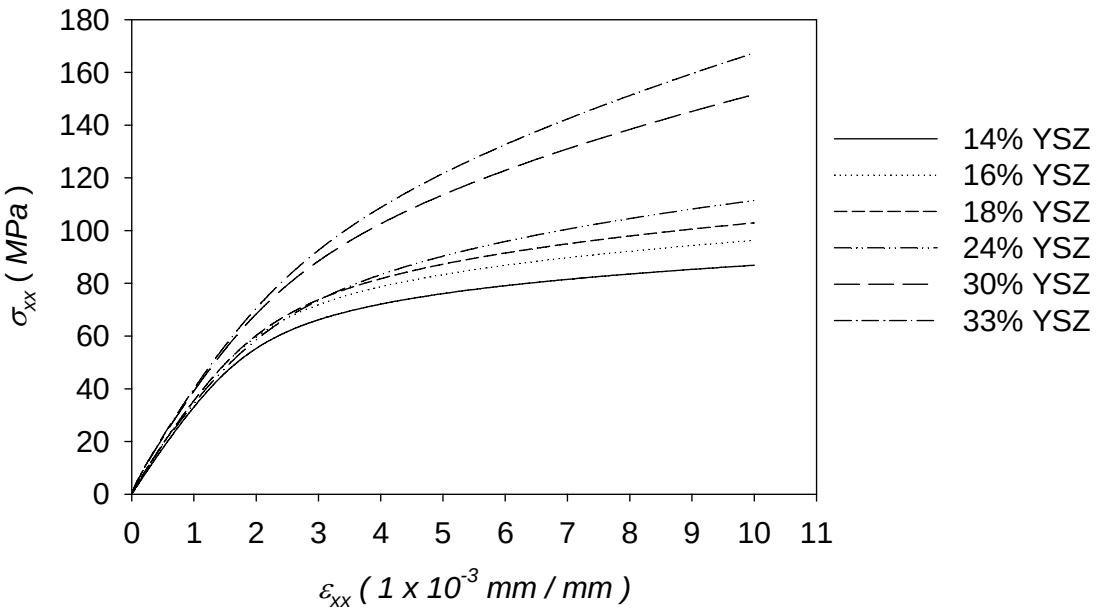


Figure 7.10. Stress-strain curves for changing volume fraction YSZ for a strain rate 1 x 10^{-6} /s at 500°C.

The impact of YSZ volume fraction is obvious from Figure 7.10, and it can be seen that at 14% and 16% YSZ, the curves almost plateau, even just for a short time. This corresponds to the prediction in Figure 3.5 that the phase will no longer percolate. The fact that some load still exists is probably an artifact of the discretized microstructure and the clustering assumption of full sides, where full sides of the cube are touching instead of edges or corners. Volume size also has an impact, since percolation threshold predicts an infinite cluster but does not prohibit a large cluster for a given volume. Table 7.3 lists the instantaneous modulus of each realization and also the steady-state value as

strain approaches infinity. The E_i value stays fairly consistent for all volume fractions, but the other values change significantly. A plot of the steady-state modulus as strain approaches infinity is the most obvious connection to the predicted percolation threshold as shown in Figure 7.11. Interestingly at a certain point above 18% this relationship also becomes linear.

Table 7.3. Composite creep values for increasing volume fraction of YSZ.

φ_{YSZ}	E_i (GPa)	σ at $\varepsilon = 1.0\times10^{-2}$ (MPa)	E_{ss} at $\varepsilon = 1.0\times10^{-2}$ (GPa)	E_{ss} at $\varepsilon \to \infty$ (GPa)
0.14	37.90	86.65	1.60	0.007
0.16	40.77	96.27	1.77	0.006
0.18	40.96	102.93	2.52	0.012
0.24	39.75	111.44	3.13	0.078
0.30	42.13	151.65	6.19	1.71
0.33	42.07	167.38	7.61	2.49

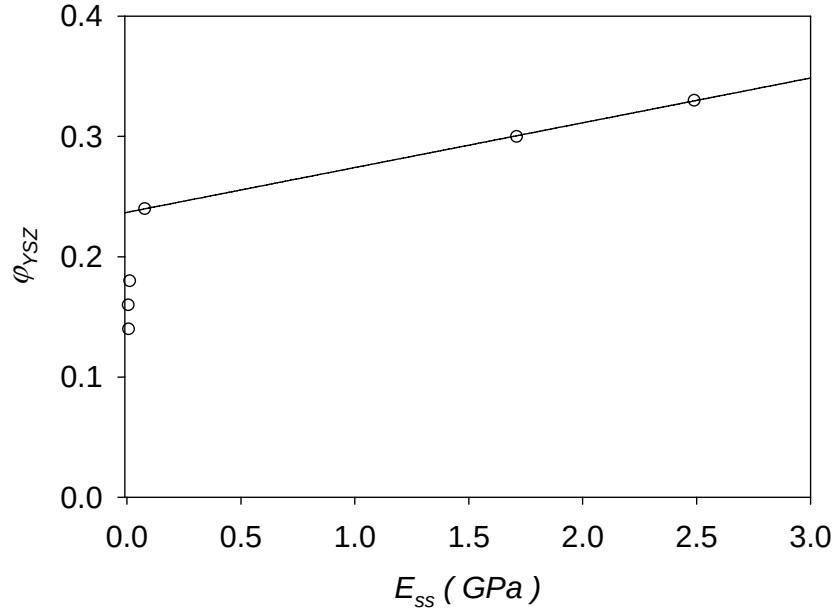


Figure 7.11. Steady-state modulus as strain approaches infinite.

7.3.4. Nickel Length Scales

To study the impact of length scales, the stress relaxation for three different models was studied at 500ºC. The first is the same realization used in Figure 7.6, but now a realization with a nickel characteristic length of 0.3 μm and 0.9 μm is also added (refer to Table 7.1). Each model was initially loaded to a pre-stress of 40MPa before allowing time-dependent deformation. The stress relaxation curves are shown in Figure 7.12.

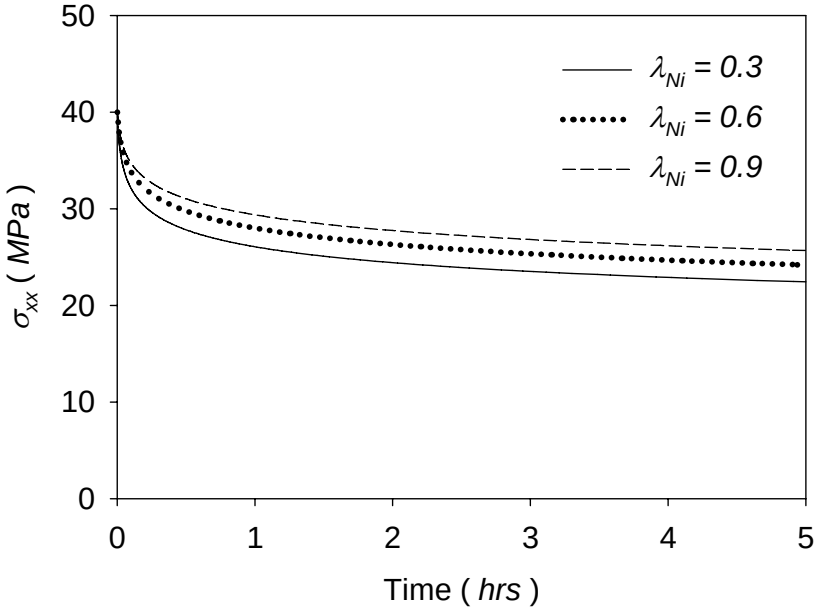


Figure 7.12. Stress relaxation for nickel length scales at 500°C.

For the realization with a smaller nickel characteristic length, the stress relaxation over time is slightly higher. In Figure 7.13 the mark correlation functions are plotted for a value two times greater than the initial stress of 40MPa. Since the nickel stresses converge towards a mean, only the YSZ function is shown and the function is normalized by YSZ's 2-point probability function. It can be seen that over time, the magnitude of the stresses decreases, but the overall shape of the curves does not change. The smaller nickel characteristic length also results in a higher stress distribution in YSZ, which explains the higher stress relaxation seen in the 0.3 μm nickel length scale. The smaller scale results in higher stresses between the two phases, increasing the amount of creep nickel experiences. This is not an immediately obvious conclusion about creep behavior in the cermet, since the first assumption would be that larger nickel sizes would lead to higher creep. Also note that for the two larger nickel length scales the difference in

behavior and mark functions is not as pronounced. This leads to the conclusion that the nickel length scale must be smaller than the YSZ and pore scales in order to produce a significant change.

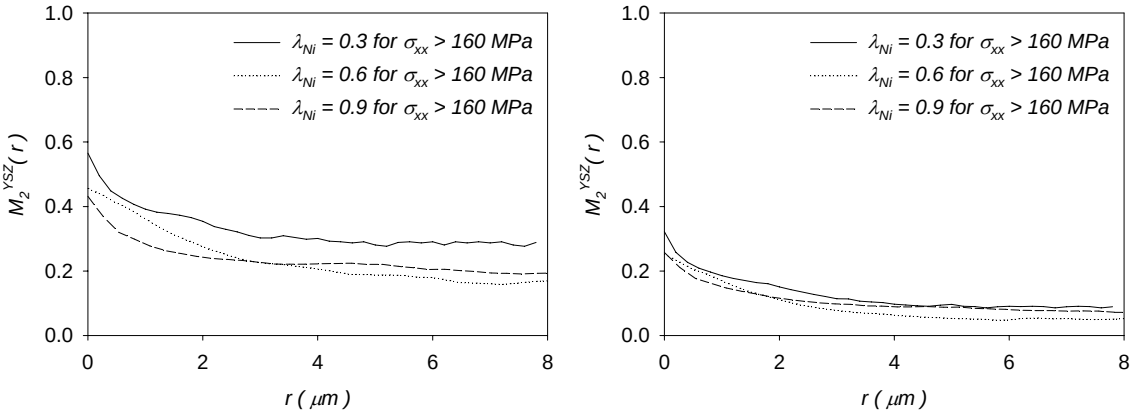


Figure 7.13 The YSZ mark function at (a) zero time and (b) 5 hours.

7.4. Summary

The preceding results show the significant impact that nickel creep has on the cermet. Data regarding the size of the RVE followed a behavior similar to that of damage and plasticity, but was difficult to determine for infinite times. However, very early on, the YSZ microstructure dominated deformation. Several factors influence creep behavior, and it was found that initial stress and operating temperature both have serious impacts. Finally, the rate of deformation and the amount of stress relaxation was affected by YSZ percolation and nickel length scales, respectively.

CHAPTER 8

SUMMARY AND CONCLUSIONS

Three-phase composites are a fundamental component of planar solid oxide fuel cells, a high density power system. In particular, the anode, the focus of this work, consists of nickel, YSZ, and a continuous pore phase, where each phase is vital to the proper operation of the fuel cell. Numerical studies of the PEN layer have found that failure will most likely occur in the anode layer since it provides the structural strength for the layer [13-16]. However, an extensive study of the anode's structural behavior in relation to its microstructure is lacking. This dissertation outlines a numerical method to study composite behavior in relation to its microstructure and seeks to improve understanding of porous cermets in general. Each chapter outlined a different component of the analysis along with results and predictions of the composite behavior. This final chapter briefly summarizes the preceding chapters, reviews major conclusions, and then discusses the significance of this research.

8.1. Chapter summary

The previous chapters have outlined a numerical methodology to study porous cermets using voxel reconstructions. The approach first created a new "realization," which was a computer generated image constructed of voxels that matched a given set of probability functions that were obtained from a physical representation of the microstructure. This realization was then used to study multiple properties for the

composite ranging from linear material properties to nonlinear deformation. Chapter One introduced the theory and methodology behind creating the microstructure realizations. Two different probability functions were used to create a realization of an ORNL anode sample, the 2-point probability and lineal path functions. Chapter Two continued to study the realizations, by using the cluster function to study whether a phase would percolate or not.

The next two chapters, four and five, studied the linear material properties of porous cermets grouped as structural or transport properties. In Chapter Four, an exhaustive study of the appropriate RVE and voxel size were undertaken in determination of Young's modulus and CTE. The methodology to determine RVE size was then used in Chapter Five to determine the acceptable RVE size for determination of thermal conductivity. The structural properties of modulus and CTE were predicted for different porosities with a good match to experimental data, and once interfacial resistivity was accounted for in the transport analysis, a good match to experimental results was also achieved. The final section of each of these chapters was the creation of new realization sets based on modified probability functions. This was done to link microstructural behavior to the probability descriptors of the microstructure.

From here, the research shifted from the prediction of material properties to a nonlinear analysis of deformation behavior. The anode can be expected to experience high stresses during assembly and from cell configuration, plus prolonged thermal stresses during operation, both of which lead to nonlinear material behavior. First, damage and plasticity were incorporated in the FE models and stress-strain curves plotted for multiple realizations. The microstructures were studied by examining the average

stresses in nickel and YSZ, determining the shape of the stress distribution within the microstructure, or finally by use of a mark function. The mark function treats stresses, or other field parameters, in the microstructure in a fashion similar to that of a probability function that describes a specific phase. The methods of data analysis provided a probabilistic description of the stresses in the composite. In Chapter Seven, time-dependent deformation was studied by adding creep behavior to the nickel phase. Since the PEN layer is known to experience stress relaxation after extended periods of time in high operating temperatures, the anode behavior was specifically studied for that situation. Also investigated were stress-strain curves for loading with a constant strain rate, and this proved to be an effective way to see the impact of YSZ percolation on the creep behavior of Ni-YSZ.

8.2. Major conclusions

Each of the preceding chapters in this work studied a specific aspect of the anode relevant to successful operation of the fuel cell and from that several conclusions about cermet behavior were drawn. Because each analysis was based on similar sets of realizations, the composite behavior can now be connected across many different material behaviors. Initially a study of percolation in Chapter 3 found that the minimum volume fraction needed for continuous porosity was 16%. It was also found the modulus was more sensitive to percolation length than to cluster size, and a larger pore percolation length gave a lower modulus. In fact, pore percolation was the only factor that could be linked across multiple changes in microstructural length scales. Plus, improvements in modulus lead to a decrease in CTE, which is beneficial to SOFCs since a common goal is to match the anode's CTE to YSZ. In Chapter Five's transport analysis, it was found that

interfacial resistance must be included in the thermal modeling, for it can contribute as much as 50% to the total effective thermal conductivity. Each effective property was found to be most influenced by a different feature of the microstructure;

- the modulus changes most with change in pore volume fraction,
- an increase in the percolation length of the porosity would cause a smaller drop in modulus,
- the CTE was independent of porosity and strongly temperature dependent,
- and thermal conductivity had a linear relationship with the interfacial area of nickel and YSZ.

For nonlinear deformations, specific conclusions were drawn from studying the interaction of nickel and YSZ. It was found that the brittle and ductile nature of YSZ and nickel, would lead to damage spreading perpendicular from the strain, while plastic strain increases the most in the direction of strain. The most significant change in post-yielding behavior came from increasing the length scale of YSZ, while increasing pore size had a minimal effect. Larger clustering of YSZ inhibited damage in that phase, and it was found that although porosity affects modulus, nonlinear deformation is controlled by the interaction of the nickel and YSZ phases.

In the second nonlinear analysis for time-dependent deformation, YSZ controls the final shape of the microstructure, but the length scale of nickel will influence the rate of stress relaxation. Both temperature and the initial stress also strongly influence the rate of deformation; however, the impact of pre-stress is greater since the initial compression of nickel, due to thermal stresses, controls the amount of nickel creep.

The results from Chapters Seven and Eight lead to the following conclusions

- the post-yielding behavior of the composite is strongly influenced by changes in the length scale of YSZ or nickel, and
- initial creep behavior is strongly influenced by nickel, but over time YSZ controls the deformation.

In a slightly different category, conclusions can also be drawn about the methodologies used in this research. For instance, especially for the linear material properties, a realization based on the 2-point probability functions accurately predicted cermet behavior. Also for linear properties, RVE size controlled standard deviation while voxel size (or discretization) controlled accuracy. This was not true for the nonlinear analysis, and the RVE size needed to be larger for such analyses than was the case for a linear analysis. The use of multiple realization sizes showed that the final results were consistent across multiple realization sets for both linear and nonlinear analyses once convergence of the RVE was reached.

8.3. Contributions

Although this research was conducted based on the anode material used in fuel cells, the methodologies developed here can be applied to other composites. The overall process served as a platform to study a complex microstructure in a wide variety of ways, and with minimal computational expense. The research modified existing

methods and developed new techniques to provide new information about three-phase composites. Tools that were enhanced include

- modification of SAM to model three-phase composites in 3D,
- use and then actual modification of realistic probability functions during the stochastic reconstruction to relate material behavior to a given microstructure, and
- the use of histograms and mark functions to study the internal stress distribution within the microstructure.

New techniques in this research were developed in several different areas ranging from RVE size determination to effective properties. They are described as follows;

- the use of discretization and RVE size parameters and box plots to provide a visual tool in the determination of convergence for FE models for effective properties,
- introduction of the percolation length to relate changes in pore size to changes in modulus and CTE,
- calculation of interfacial resistivity parameter to account for imperfect interfaces in the FE model, and to accurately predict thermal conductivity, and
- the study of YSZ percolation and introduction of a steady-state modulus in examining creep in the Ni-YSZ cermet.

Overall the research met the requirements for a successful analysis of three-phase cermets. It modeled the interpenetrating microstructures in three dimensions, while capturing the percolating nature of all three phases. Also important was the ability to study the stress distributions occurring within the microstructure. To study the internal stresses in this way allowed conclusions to be drawn about the interactions between

phases and resulting bulk behavior. To that end some significant contributions concerning cermet behavior are

- changes in porosity are more significant to modulus, than changes in internal length scales
- changes in length scales of the microstructure will more strongly influence post-yield and creep behavior than either modulus or CTE,
- the interface of nickel and YSZ is a significant factor for transport properties, and
- the initial stress in the cermet will have a larger impact on stress relaxation than temperature.

8.4. Future work

Further work in this research can take place down several avenues. The accuracy of voxel reconstructions could be studied for new microstructures, or modified microstructures could be used to test the realizations' ability to predict material behavior. Specific to fuel cell research, further study of post-yielding and fatigue behavior can provide more insights into failure of the PEN layer. Introduction of an incremental plasticity model in nickel would allow the accurate modeling of unloading behavior. First, experimental studies need to be developed to investigate the specific behavior of nickel, and to a lesser extent YSZ, within the anode. This would lead to more accurate material data to improve the accuracy of the analysis. Still voxel reconstructions proved to be an effective way to study three-phase composites and have potential for further development.

APPENDIX A

PROBABILITY INDEPENDENCE

The voxel reconstruction used $S_2^{(1)}$, $S_2^{(2)}$, and $S_2^{(3)}$ to recreate each realization. The following formulation, based on a personal communication from Dr. Garmestani at Georgia Tech [59], proves that the three probability functions completely describe the microstructure.

Global normality requires that

$$\sum_{i=1}^{3}\sum_{j=1}^{3} S_2^{(ij)} = 1 \tag{A.1}$$

Local normality requires that

$$\sum_{j=1}^{3} S_2^{(ij)} = V_i \tag{A.2}$$

Equations (A.1) and (A.2), provide a complete relationship between the following probability functions. Accounting for symmetry of the probability functions, if the three functions used result in an independent matrix then the microstructure is accurately described.

For the set of $S_2^{(1)}$, $S_2^{(2)}$, and $S_2^{(3)}$ the matrix takes the form in (A.3).

$$A = \begin{bmatrix} S_2^{(11)} & 1 & 1 \\ 1 & S_2^{(22)} & 1 \\ 1 & 1 & S_2^{(33)} \end{bmatrix} \tag{A.3}$$

Since the determinant of A is greater than zero the solution to the set of equations is nontrivial. However, for $S_2^{(11)}$, $S_2^{(12)}$, and $S_2^{(22)}$ the determinant is equal to zero and these probability functions do no describe the microstructure, as shown in (A.4).

$$\det \begin{vmatrix} S_2^{(11)} & S_2^{(12)} & 1 \\ S_2^{(12)} & S_2^{(22)} & 1 \\ 1 & 1 & 1 \end{vmatrix} = 0 \tag{A.4}$$

APPENDIX B

RECONSTRUCTION CODE

Table A.1 provides a list of the CPP code used for the voxel reconstruction. The table starts with the base program, followed with the environment header file. Each following row lists the subprograms and header files listed in environment.h.

Table A.1. Reconstruction code.

createMatDebye.cpp

```
#include "environment.h"

int createMat( FileCall&, const SIZE&, vector<Material>&, bool, double, int );

int main()     {
        string dir = "//nv//hp4//gtg580j//c_files//datFiles//";
        std::clock_t start = std::clock();
        Material phase1("NI", .35, 1), phase2("YSZ", .43, 2);
        Material phase3 = --(phase1 + phase2);
        vector <Material> phases;
        phases.push_back(phase1);
        phases.push_back(phase2);
        phases.push_back(phase3);
        cout <<"The microstructure is: " <<endl;
        cout <<phase1 <<endl <<phase2 <<endl <<phase3  <<endl;

        SIZE INN(40);
        int numReal = 10;
        queryDefaults( INN, numReal );
        string groupName = convert( INN.W() );

        cout <<"Current identifier is " <<groupName <<endl;
        groupName = groupName + queryString();

        double cl;
        cout <<"What is diameter " <<endl;
        cin >>cl;
        int lro;
        cout <<"What is the LRO " <<endl;
        cin >>lro;

        string labelStr;
        for ( int i = 0; i < numReal; i++ )   {
                labelStr = "seq" + convert( i + 1 );
                FileCall names( dir, "ranMat", groupName, labelStr, ".dat" );
                srand( i * 10 );
                createMat(names, INN, phases, true, cl, lro);
        }

        cout <<"Clock time: ";
        cout <<(( std::clock()- start)/(double)CLOCKS_PER_SEC ) <<'\n';
        return 0;  }

int createMat( FileCall& fc, const SIZE& INN, vector<Material>& phases, bool third,
double cl, int lro  )
{
        string name = fc.name();
        cout <<"Current file name: " <<name <<endl <<endl;

        FileCall record(fc.getDir(), "log", fc.getDes(), "", ".log" );
        string RECORD = record.name();

        //LOG FILE AND INITIALIZE CLOCK TIME
        ofstream log( RECORD.c_str(), ios::app );
        header( log, name, phases, INN );
        log << "The diameter is " <<cl <<endl;
```

```
log << "The long range order is " <<lro <<endl;
std::clock_t start = std::clock();


//CREATE SAMPLE
const SIZE OUT = (INN > 10) ? INN + MAX_BORDER : INN + MIN_BORDER;
int sample[OUT.S()];
fillArray(sample, OUT.S(), 1);

vector<Material>::iterator iter = phases.begin();
++iter;
while ( iter != phases.end() ) {
        fillCenter(sample, OUT, INN, *iter);
        fillBorder(sample, OUT, INN, *iter);
        ++iter;        }

iter = phases.begin();
while ( iter != phases.end() )        {
        log <<*iter;
        ++iter; }

log <<OUT <<endl <<INN;
//Ratio of CL's
double ratioCharLen = 0.66666667;// is ORNL value
log <<"The ratio of char. lengths is " <<ratioCharLen <<endl;

//STOCHASTIC INITIALIZATION
{
int radius = ( int ) ( 2 * lro );
//Stochastic <Corr Function> name(material, SIZE, radius, per. overlap);
Stochastic <TwoPoint> stoch1(phases[0], OUT, radius, (int)cl);
Stochastic <TwoPoint> stoch2(phases[1], OUT, radius, (int)(cl));
Stochastic <TwoPoint> stoch3(phases[2], OUT, radius, (int)(cl));
Energy s1( cl, phases[0], 'D' );
Energy s2( ratioCharLen * cl, phases[1], 'D' );
Energy s3( ratioCharLen * cl, phases[2], 'D' );
stoch1.total ( sample );
stoch2.total ( sample );
stoch3.total ( sample );

double eng = 0; double engPrime = 0;
eng = s1.calc( stoch1.funcOut() ) + s2.calc( stoch2.funcOut() );
double factor = 1;

if ( third == true )   {
        stoch3.total ( sample );
        eng = eng + factor * s3.calc( stoch3.funcOut() );    }

//START LOOP
Counter c, reject, accept;                    //Initialize counters
double acceptRate = 0;
Schedule deluge( THRESHOLD );                 //Great Deluge algorithm
IdLoc pA(3, OUT, INN), pB(3, OUT, INN);
Coord cA = pA.get_loc();      Coord cB = pB.get_loc();
VolStatus status;

while ( eng > ERROR )  {
        c.incCount();

        pA.rotate();
        pB.rotate();

        std::clock_t startInt = std::clock();

        if ( status.get() == true )
                findOnInterface( sample, pA, pB );
        else
                correctVolStatus( sample, pA, pB, status );

        cA = pA.get_loc();     cB = pB.get_loc();
```

```
                double endInt = (std::clock() - startInt)/(double)CLOCKS_PER_SEC;

                std::clock_t startStoch = std::clock();
                stoch1.temp( sample, cA, cB );
                stoch2.temp( sample, cA, cB );

                engPrime = s1.calc( stoch1.funcOut() ) + s2.calc( stoch2.funcOut() );
                //engPrime = s1.calc( stoch1.funcOut() );
                if ( third == true )   {
                        stoch3.temp( sample, cA, cB );
                        engPrime = engPrime + factor * s3.calc( stoch3.funcOut() );  }

                if ( deluge.deluge(eng, engPrime) == 1 || reject == INN.S() / 2 ) {
                        eng = engPrime;
                        stoch1.accept(cA, cB); stoch2.accept(cA, cB);
                        if ( third == true ) stoch3.accept(cA, cB);
                        reject.reset();accept.incCount();
                        if ( status.get() == false )
                                status.reset();
                        else
                                status.check(pA, pB);
                }
                else    {
                        pA.resetId( sample ); pB.resetId( sample );
                        reject.incCount();
                        stoch1.reject();       stoch2.reject();
                        if ( third == true ) stoch3.reject();
                }

                double endStoch = (std::clock() - startStoch)/(double)CLOCKS_PER_SEC;

                if ( c % STATUS_COUNT == 0 )   {
                        acceptRate = accept / STATUS_COUNT * 100;
                        currentStatus( cout, eng, c, reject, acceptRate);
                        currentStatus( log, eng, c, reject, acceptRate);
                        accept.reset();
                        //cout <<"\t\t" <<endStoch <<"\t\t" <<endInt <<endl;
                        }

                if ( c > MAX_COUNT )   break;
        }
        finalStatus( cout, eng, c, (( std::clock()- start)/(double)CLOCKS_PER_SEC ));
        finalStatus( log, eng, c, (( std::clock()- start)/(double)CLOCKS_PER_SEC ));
        }
        //SHRINK ARRAY
        int center[INN.S()];
        shrinkArray(sample, center, INN, OUT);
        dataOutput(center, INN, phases, name);
        return 0;}
```

environment.h

```
#include "jmath.h"
#include "random.h"
#include "arrayFunctions.h"
#include "dataFiles.h"

#include "Counter.h"
#include "SIZE.h"
#include "Coord.h"
#include "Material.h"
#include "Stochastic1.h"
#include "IdLoc.h"
#include "VolStatus.h"
#include "Schedule.h"
#include "InputPar.h"

#include <ctime>

#include <string>
using std::string;
```

```
#include <fstream>
using std::ifstream;
using std::ofstream;

#include <iostream>
using std::cout;
using std::cin;
using std::endl;
using std::ios;

#include <iomanip>
using std::setw;
using std::setprecision;

#include <vector>
using std::vector;
using std::iterator;
const double ERROR = 1e-7;
const double THRESHOLD = .000001;
const int MAX_BORDER = 0;
const int MIN_BORDER = 0;
const int MAX_COUNT = 10000000;
const int STATUS_COUNT = 10000;
```

jmath.h

```
#ifndef GUARD_jmath_h
#define GUARD_jmath_h

#include "random.h"
#include "SIZE.h"
#include "Coord.h"
#include "arrayFunctions.h"
#include <cmath>
#include <vector>
using std::vector;
#include <iostream>
using std::cout;
const double PI = acos(-1.0);

double heaviside(double);
double overlapping_sphere(double, double, double);
double debye_oscill(double, double, double, double);
double debye_decay(double, double, double);
double poly(long double, long double, long double, long double, long double, long double,
long double, long double);
int bin( double val );
vector <double> pureTwoPoint( int, SIZE, int* );
vector <double> pureTwoPoint( int, int, SIZE, int* );
#endif
```

jmath.cpp

```
#include "jmath.h"

double heaviside(double x)
{
        if ( x > 0 )
                return 1;
        else if ( x == 0 )
                return .5;
        else
                return 0;
}
double overlapping_sphere(double dia, double r, double vf)
{
        double g_func;
        double o_func;
        double vf2 = (double) 1 - vf;

        g_func = pow(dia, 2) / 2 *
                        ( PI - heaviside(dia - r) *
                        ( acos( r / dia ) - (r / dia) * sqrt( fabs(1 - pow (r / dia,
2)))));
```

```
        o_func = 1 - 2 * vf2 + pow( vf2 , 4 * g_func / ( PI * pow(dia, 2)));

        return o_func;
}
double debye_oscill(double a, double b, double r, double vf)
{
        double o_func;
        double vf2 = (double) 1 - vf;
        double q = 2 * PI / b;
        if (r == 0)
                o_func = vf;
        else
                o_func = vf * vf2 * exp ( -r / a ) * sin ( q * r ) / ( q * r ) + pow( vf,
2 );
        return o_func;
}
double debye_decay(double charLength, double r, double vf)
{
        double o_func;

        o_func = (1-vf) * vf * exp ( -r / charLength ) + vf * vf;

        return o_func;
}
double poly(long double c1, long double c2, long double c3, long double c4, long double
c5, long double c6, long double c7, long double x)

{
        double o_func;

        o_func = c1 * pow( x, 6 ) + c2 * pow( x, 5 ) + c3 * pow( x, 4 ) + c4 * pow( x, 3 )
+ c5 * pow( x, 2 ) + c6 * x + c7;
        return o_func;
}
int bin( double val )  {
        double decimal = val - (int) val ;
        if ( decimal >= .5 )
                return (int)ceil( val );
        else
                return (int)floor( val );
}

vector <double> pureTwoPoint ( int mat, SIZE d, int* arr )   {
        int maxR = d.min();
        double curR = 0;
        int binR = 0;

        int trials[maxR];
        int hits[maxR];
        double func[maxR];

        for ( int i = 0; i < maxR; ++i )      {
                trials[i] = 0;
                hits[i] = 0;
                func[i] = 0;   }

        int ct = 0;
        while ( ct < 1000 )    {
                Coord a, b;
                a.set( random( d.W() ), random( d.H() ), random( d.D() ) );
                b.set( random( d.W() ), random( d.H() ), random( d.D() ) );
                curR = distance( a, b );
                binR = bin( curR );
                if ( binR < maxR )     {
                        trials[ binR ]++;
                        if ( arr[ index( a, d ) ] == mat && arr[ index( b, d ) ] == mat ) {
                                hits[ binR ]++;
                                func[ binR ] = (double)hits[ binR] / (double)trials[ binR
];      }
                        if ( binR == 0 )
                                ct++;                 }
```

```
                if ( ct % 10 == 0 )    {
                        //cout <<"Current count " <<ct <<endl;
                        ct++;   }
        }

        vector<double> temp;
        for ( int i = 0; i < maxR; ++i )
                temp.push_back( func[i] );

        return temp;
}
vector <double> pureTwoPoint ( int mat1, int mat2, SIZE d, int* arr )       {
        int maxR = d.min();
        double curR = 0;
        int binR = 0;

        int trials[maxR];
        int hits[maxR];
        double func[maxR];

        for ( int i = 0; i < maxR; ++i )      {
                trials[i] = 0;
                hits[i] = 0;
                func[i] = 0;   }

        int ct = 0;
        while ( ct < 100000 )  {
                Coord a, b;
                a.set( random( d.W() ), random( d.H() ), random( d.D() ) );
                b.set( random( d.W() ), random( d.H() ), random( d.D() ) );
                curR = distance( a, b );
                binR = bin( curR );
                if ( binR < maxR )     {
                        trials[ binR ]++;
                        if ( arr[ index( a, d ) ] == mat1 && arr[ index( b, d ) ] == mat2 )
{
                                hits[ binR ]++;
                                func[ binR ] = (double)hits[ binR] / (double)trials[ binR
];      }
                        if ( binR == 0 )
                                ct++;   }
                if ( ct % 100 == 0 )   {
                        cout <<"Current count " <<ct <<endl;
                        ct++;   }
        }

        vector<double> temp;
        for ( int i = 0; i < maxR; ++i )
                temp.push_back( func[i] );
        return temp;}}
```

random.h

```
#ifndef GUARD_random_h
#define GUARD_random_h
#include <cstdlib>
#include <iostream>
#include <fstream>
using std::ostream;
using std::cout;
using std::endl;


//random.......sends back random value between 0 and b
//random.......sends back random value between a and b
//NOTE:  MIN VALUE EQUAL TO a OR 0
//NOTE:  MAX VALUE 1 LESS THAN b
//randomDouble. sends back double value between a and b
//randomSeed...seeds random numbers
//NOTE:  SOURCE CODE FROM ETTER C++ BOOK

int random( int value );
```

```
int random( int a, int b );
double randomDouble( double a, double b );
void randomSeed(int value );

#endif
```

random.cpp

```
// random.cpp

#include "random.h"

int random( int value )
{
	return rand() % value;
}

int random( int a, int b )
{
	return rand() % ( b - a + 1 ) + a;
}

double randomDouble( double a, double b )
{
	return ((double)rand()/RAND_MAX)*(b - a) + a;
}

void randomSeed( int value )
{
	srand( value );
}
```

arrayFunctions.h

```
#ifndef GUARD_arrayFunctions_h
#define GUARD_arrayFunctions_h
#include "random.h"
#include "SIZE.h"
#include "Coord.h"
#include "Material.h"
#include <cmath>
#include <iomanip>
using std::setprecision;
#include <iostream>
using std::cout;
using std::endl;
#include <fstream>
using std::ofstream;

//fillArray... initializes 1D array with input value
//fillSpace... inputs value at desired 3D coordinates in 1D array
//                              overloaded
//fillCenter.. randomly distributes input value throughout center of 1D array
//fillBorder.. randomly distributes input value throughout border of 1D array
//count........counts number of given input value in 1D array
//screen.......prints 1D array as 3D array
//                              x -> rows, y -> columns, z -> each block
//                              starts counting at zero
//screenRow... prints all rows for given y and z
//screenCol... prints all columns for given x and z
//screenDep... prints all depths for given x and y
//refineArray. subdivides 1D array ( 1 grid to 4 )
//shrinkArray. given inner and outer borders creates new 1D array of inner size

void fillArray(int*, const int, int);
void fillSpace(int*, const SIZE, int, int, int, int);
void fillSpace(int*, const SIZE, int, Coord);
void fillCenter(int* arr, const SIZE, const SIZE, Material mat );
void fillBorder(int* arr, const SIZE, const SIZE, Material mat );
int count(int * arr, const SIZE s, int id);
int countBorder(int* arr, const SIZE, const SIZE, int);
void screen(int * arr, const SIZE);
void screenRow(int * arr, const SIZE, int col, int dep);
```

```
void screenCol(int * arr, const SIZE, int, int);
void screenDep(int * arr, const SIZE, int, int);
void outputFile(int * arr, const SIZE, string );
//void readFile(string);
void refineArray(int* old, int* new_, const SIZE);
void shrinkArray(int* old, int* new_, const SIZE&, const SIZE&);
#endif
```

arrayFunctions.cpp

```
#include "arrayFunctions.h"

int main()
{
	{
	//SET UP CONDITIONS FOR USE
	const SIZE out(30, 4, 4);
	const SIZE in(20, 2, 2);
	cout <<out <<in;

	Material m1("ONE", .5, 6);
	Material m2("TWO", .4, 7);
	Material m3 = --(m1 + m2);
	cout <<m1 <<m2 <<m3;
	cout <<endl;

	//INITIALIZE ARRAY
	int arr[out.S()];

	//TEST OF FILL ARRAY
	cout <<"TEST OF FILL ARRAY";
	fillArray(arr, out.S(), 1);
	screen(arr, out);

	//TEST OF FILL CENTER OF ARRAY
	cout <<"TEST OF FILL CENTER";
	fillCenter(arr, out, in, m1);
	fillCenter(arr, out, in, m2);
	screen(arr, out);

	//TEST OF FILL BORDER OF ARRAY
	cout <<"TEST OF FILL BORDER";
	fillArray(arr, out.S(), 1);
	fillBorder(arr, out, in, m1);
	fillBorder(arr, out, in, m2);
	fillBorder(arr, out, in, m3);
	screen(arr,out);

	//REFILL CENTER OF ARRAY
	cout <<"REFILL CENTER";
	fillCenter(arr, out, in, m1);
	fillCenter(arr, out, in, m2);
	fillCenter(arr, out, in, m3);
	screen(arr, out);

	//TEST COUNT OF EACH ID
	cout <<"TEST COUNT OF EACH ID" <<endl;
	cout <<m1.get_id() <<": " <<count(arr, out, m1.get_id()) <<endl;
	cout <<m2.get_id() <<": " <<count(arr, out, m2.get_id()) <<endl;
	cout <<m3.get_id() <<": " <<count(arr, out, m3.get_id()) <<endl;
	cout <<1 <<": " <<count(arr, out, 1) <<endl;

	//TEST SCREEN PRINTOUTS OF EACH ROW COL AND DEP
	cout <<"TEST SEPERATE SCREEN PRINT OUTS" <<endl;
	cout <<"(0, 0, 1)" <<endl;
	cout <<"ROW:  " <<endl;
	screenRow(arr, out, 0, 1);
	cout <<"COL:  " <<endl;
	screenCol(arr, out, 0, 1);
	cout <<"DEP:  " <<endl;
	screenDep(arr, out, 0, 0);
```

```
        //TEST SHRINK ARRAY
        cout <<endl <<"TEST SHRINK ARRAY";
        int arr_shrink[in.S()];
        shrinkArray(arr, arr_shrink, out, in);
        screen(arr_shrink, in);

        //TEST REFINE ARRAY
        cout <<"REFINE ARRAY";
        SIZE large = in * 2;
        int arr_ref[large.S()];
        refineArray(arr, arr_ref, in);
        screen(arr_ref, large);
        }
        //TEST ERROR OUTPUT FOR FILL CENTER
        cout <<"TEST ERROR OUTPUT FOR FILL CENTER" <<endl;
        const SIZE out(30, 4, 4);
        const SIZE in(20, 2, 2);
        cout <<out <<in;

        Material m1("ONE", .1, 6);
        Material m2 = --m1;
        cout <<m1 <<m2;
        cout <<endl;

        //INITIALIZE ARRAY
        int arr[out.S()];
        //TEST OF FILL ARRAY
        fillArray(arr, out.S(), 1);

        //TEST OF FILL CENTER ERROR OF ARRAY
        cout <<"TEST OF ERROR FOR FILL CENTER";
        fillCenter(arr, out, in, m1);
        fillCenter(arr, out, in, m2);
        screen(arr, out);

        //TEST OF FILL BORDER ERROR OF ARRAY
        cout <<"TEST OF ERROR FOR FILL BORDER";
        fillBorder(arr, out, in, m1);
        fillBorder(arr, out, in, m2);
        screen(arr, out);

        return 0;}
```

dataFiles.h

```
#ifndef GUARD_dataFiles_h
#define GUARD_dataFiles_h
#include "SIZE.h"
#include "Material.h"
#include "Counter.h"
#include "Stochastic1.h"
#include "SurfArea.h"
#include "InputPar.h"
#include "jmath.h"
#include <ctime>
#include <vector>
using std::vector;
#include <sstream>
#include <string.h>
using std::string;
#include <iomanip>
using std::setw;
using std::setprecision;
#include <iostream>
using std::cerr;
using std::cout;
using std::endl;
using std::cin;
using std::ios;
#include <fstream>
using std::ifstream;
using std::ofstream;
```

```
void ansysOutput( int*, const SIZE, const char*, char );
       //Output data to ANSYS Element file
       //Requires input ANSYS element file
void dataOutput( int*, const SIZE&, const vector<Material>&, string );
void dataOutputTwoPhase( int*, const SIZE&, const vector<Material>&, string );
int setSize ( const char* );
void dataSort( double*, SIZE& , const char*);
void dataInput( double*, const char* file_name );
void dataInput( int*, SIZE&, vector<Material>&, const char* file_name );
void dataInput( SIZE& s, vector<Material>& p, const char* file_name );
ostream& header( ostream&, string, vector<Material>, SIZE );
ostream& currentStatus( ostream& out, double eng, Counter& total, Counter& reject, double
rate);
ostream& finalStatus( ostream& out, double eng, Counter& total, double time );
ostream& stochFuncs( ostream& , vector<Material>, SIZE, int*, double*** );
ostream& stochStat( ostream& out, double*** record, int n, int type, int rad );
ostream& stochHeader(ostream&, vector<Material> );
ostream& areaData( ostream&, vector<Material>, SIZE, string, int*, vector<vector<int> >&
);
ostream& areaStat( ostream& out, vector<vector<int> >& record);
ostream& areaHeader( ostream&, vector<Material> );
ostream& leastSquare( ostream&, vector<Material>, double cl, const SIZE&, double*** );
#endif
```

dataFiles.cpp

```
#include "dataFiles.h"
void ansysOutput(int* arr, const SIZE dim, const char* file_name, char file_in[40])
{
       int count=0, row = 0, col = 0, dep = 0;
       int j=0, k=1, l=2, m=3, n=4, o=5;
       int p=6, q=7, r=8, s=9, t=10, u=11, v=12, w=13;

       ofstream file_new(file_name);
       ifstream elemdata; // indata is like cin
       int data1; // variable for input value

       elemdata.open(file_in); // opens the file
       if(!elemdata) { // file couldn't be opened
     cerr << "Error: file could not be opened" << endl;
     exit(1); }

       elemdata >> data1;
       while ( !elemdata.eof() ) { // keep reading until end-of-file
              if (count==j) {file_new << data1 <<",";  j=j+14;
}
              if (count==k) {file_new << data1 <<",";  k=k+14;        }
              if (count==l) {file_new << data1 <<",";  l=l+14;        }
              if (count==m) {file_new << data1 <<",";  m=m+14;        }
              if (count==n) {file_new << data1 <<",";  n=n+14;        }
              if (count==o) {file_new << data1 <<",";  o=o+14;        }
              if (count==p) {file_new << data1 <<",";  p=p+14;        }
              if (count==q) {file_new << data1 <<",";  q=q+14;        }
              if (count==r) {file_new <<arr[row * dim.H() * dim.D() + col * dim.D() +
dep] <<",";  r=r+14;  }
              if (count==s) {file_new << data1 <<",";  s=s+14;        }
              if (count==t) {file_new << data1 <<",";  t=t+14;        }
              if (count==u) {file_new << data1 <<",";  u=u+14;        }
              if (count==v) {file_new << data1 <<",";  v=v+14;        }
              if (count==w) {
                file_new << data1 <<"," <<endl;
                w=w+14;  row++;
                if ( row == dim.W() )        {
                       row = 0;  col++;
                       if ( col == dim.H() )        {
                              col = 0;  dep++;     }      }      }
         count++;
     elemdata >> data1; // sets EOF flag if no value found
       }
       elemdata.close();   file_new.close();
}
```

```
void dataOutput(int* arr, const SIZE& dim, const vector<Material>& p, string file)
{
	ofstream out(file.c_str());
	out <<dim.W() <<", " <<dim.H() <<", " <<dim.D() <<endl;
	out <<p.size() <<endl;
	for ( int i = 0; i < p.size(); i++ ) {
		out <<p[ i ].get_name() <<endl;
		out <<p[ i ].get_vf() <<endl <<p[ i ].get_id() <<endl;	}
	out <<endl <<endl;
	for ( int i = 0; i < dim.S(); i++ )
		out <<arr[ i ] <<endl;
	out.close();
}
void dataOutputTwoPhase(int* arr, const SIZE& dim, const vector<Material>& p, string
file)
{
	ofstream out(file.c_str());
	out <<dim.W() <<", " <<dim.H() <<", " <<dim.D() <<endl;
	out <<2 <<endl;
	out <<p[0].get_name() <<endl;
	out <<p[0].get_vf() <<endl;
	out <<p[0].get_id() <<endl;
	out <<"Pore" <<endl;
	out <<p[1].get_vf() <<endl;
	out <<3 <<endl;
	out <<endl <<endl;
	for ( int i = 0; i < dim.S(); i++ )   {
		if ( arr[ i ] == 2 )
			out <<3 <<endl;
		else
			out <<arr[ i ] <<endl; }
	out.close();
}
int setSize( const char* file_name )  {
	ifstream indata;
	indata.open( file_name );
	if(indata.fail()) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
      exit(1); }
	  SIZE s;
	indata >> s;
	indata.close();
	return s.S();
}
void dataSort( double* arr, SIZE& s, const char* file_name)
{
	ifstream indata;
	double var;
	double data[ s.S() ];
	indata.open(file_name);
	cout <<file_name <<endl;
	if(!indata)    {
		cerr << "Error : file could not be opened in dataSort" <<endl;
		exit(1);	}
	int i = 0;
	while ( indata >> var )	{
		data[i] = var;
		i++;		}
	i = 0;
	cout <<"total in var is " <<i <<endl;
	for (int n = 0; n < s.S(); n++)
		arr[n]=data[n];
	indata.close();
}
void dataInput( double* arr, const char* file_name )
{
	ifstream indata; // indata is like cin
	double data;
	indata.open(file_name); // opens the file
```

```
        if(!indata) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
      exit(1); }
        int i = 0;
        while (indata >> data) {
                *(arr + i) = data;
                i++;    }
        indata.close();
}
void dataInput( int* arr, SIZE& s, vector<Material>& p, const char* file_name )
{
        ifstream indata; // indata is like cin
        int data;
        indata.open(file_name); // opens the file
        if(!indata) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
      exit(1); }
        indata >> s;
        int n  = 0;
        indata >> n;
        Material temp;
        vector<Material> temp2( n );
        for ( int i = 0; i < n; i++ )  {
                indata >> temp;
                temp2[i] = temp;       }
        p = temp2;
        int i = 0;
        while (indata >> data) {
                *(arr + i) = data;
                i++;    }
        indata.close();
}
void dataInput( SIZE& s, vector<Material>& p, const char* file_name )
{
        ifstream indata; // indata is like cin
        int data;
        indata.open(file_name); // opens the file
        if(!indata) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
      exit(1); }
        indata >> s;
        int n  = 0;
        indata >> n;
        Material temp;
        vector<Material> temp2( n );
        for ( int i = 0; i < n; i++ )  {
                indata >> temp;
                temp2[i] = temp;       }
        p = temp2;
        indata.close();
}
ostream& header( ostream& out, string name, vector<Material> p, SIZE d )    {
        out <<"Current file name: " <<name;
        time_t rawtime;         struct tm * timeinfo;
        time ( &rawtime );      timeinfo = localtime ( &rawtime );
        out <<endl <<"Initial local time and date: "  <<asctime (timeinfo);
        std::clock_t start = std::clock();
        out <<"Materials" <<endl;
        for ( int i = 0; i < p.size(); i++ )
                out <<p[ i ] <<endl;
        out <<"Dimensions" <<endl;
        out <<d;
        out <<endl;
}
ostream& currentStatus( ostream& out, double eng, Counter& total, Counter& reject, double
rate)
{
        out <<"E = " <<setw(10) <<setprecision(3) <<eng;
        out <<", c = " <<setw(8) <<total <<", r = " <<setw(5) <<reject;
        out <<", AR = " <<setw(4) <<setprecision(2) <<rate <<"%" <<endl;
        return out;
```

```
}
ostream& finalStatus( ostream& out, double eng, Counter& total, double time )
{
	out <<endl;
	out <<"The final energy is " <<setprecision(3) <<eng <<" at " <<total <<"."
<<endl;
	out <<"The total Clock time was " <<time <<"." <<endl;
	return out;
}

ostream& stochFuncs( ostream& out, vector<Material> p, SIZE d, int* grid, double***
record) {
	static int staticCt(0);

	int ct = 0;
	vector<double> t;
	vector<Material>::iterator itMat = p.begin();
	while ( itMat != p.end() )	{
		//Initialize
		Stochastic<TwoPoint> twoPt(*itMat, d, d.W());		twoPt.total( grid );
		cout <<".";
		Stochastic<LinealChord> linCd(*itMat, d, d.W());	linCd.total( grid );
		cout <<".";
		Stochastic<ClusterFunc> clusFunc(*itMat, d, d.W());	clusFunc.total( grid
);
		//Stochastic<MixedPhase> clusFunc((itMat == --p.end()) ? p.front() :
*(itMat + 1), *itMat, d, d.W());
		clusFunc.total(grid);
		cout <<".";
		Stochastic<MixedPhase> mixPh(*itMat, (itMat == --p.end()) ? p.front() :
*(itMat + 1), d, d.W());
		//mixPh.setMat( *itMat /*(itMat == --p.end()) ? p.front() : *(itMat + 1)*/
);
		mixPh.total( grid );
		cout <<".";
		t = pureTwoPoint(p[ ct ].get_id(), d, grid);

		//Input into output
		for ( int i = 0; i < d.W(); i++ )	{
			record[ staticCt ][ ct * 5 + 0 ][ i ] = twoPt.get_func( i );
			record[ staticCt ][ ct * 5 + 1 ][ i ] = t[ i ];
			record[ staticCt ][ ct * 5 + 2 ][ i ] = clusFunc.get_func( i );
			record[ staticCt ][ ct * 5 + 3 ][ i ] = linCd.get_func( i );
			record[ staticCt ][ ct * 5 + 4 ][ i ] = mixPh.get_func( i ); }
		ct++;
		++itMat;	}

	for ( int j = 0; j < d.W(); j++ )	{
		out <<j <<", ";
		for ( int i = 0; i < p.size() * 5; i++ )
			out <<setprecision(3) <<record[staticCt][i][j] <<", ";
		out <<endl;	}

	staticCt++;
	return out;
}
ostream& stochStat( ostream& out, double*** record, int n, int type, int rad )	{

	double ave, aveDev, stDev, var, skew, curt;
	vector<double> dummy( n );

	out <<"average" <<endl;
	for ( int k = 0; k < rad; k++ )	{
		for ( int j = 0; j < type; j++ )	{
			for ( int i = 0; i < n; i++ )
				dummy[i] = record[i][j][k];
			statData ( dummy, ave, aveDev, stDev, var, skew, curt);
			if ( j == 0 )
				out <<k <<", ";
			out <<setprecision(3) <<ave <<", ";
		}
```

```
                out <<endl;     }

        out <<endl <<endl <<"standard deviation" <<endl;
        for ( int k = 0; k < rad; k++ )         {
                for ( int j = 0; j < type; j++ )        {
                        for ( int i = 0; i < n; i++ )
                                dummy[i] = record[i][j][k];
                        statData ( dummy, ave, aveDev, stDev, var, skew, curt);
                        if ( j == 0 )
                                out <<k <<", ";
                        out <<setprecision(3) <<stDev <<", ";
                }
                out <<endl;     }
        return out;
}

ostream& stochHeader(ostream& out, vector<Material> p)        {
        vector<Material>::iterator itMat = p.begin();
        string dummy, dummy1;
        out <<"Radius" <<", " ;
        for (int i = 0; i < p.size(); i++ ){
                dummy = p[i].get_name();
                dummy1 = ( i == p.size() - 1 ) ? p[0].get_name() : p[i + 1].get_name();
                if ( dummy.size() > 3 )
                        dummy = dummy.substr(0, 2);
                if ( dummy1.size() > 3 )
                        dummy1 = dummy1.substr(0, 2);
                out <<dummy <<", " <<dummy <<", " <<dummy <<", " <<dummy <<", "  <<dummy +
dummy1 <<", "; }
        out <<endl <<"Type" <<", ";
        itMat = p.begin();
        while ( itMat != p.end() )      {
                out <<"TwoPt" <<", " <<"Pure" <<", " <<"Clus" <<", " <<"LinCd" <<", "
<<"MxPh" <<", ";
                ++itMat;        }
        out <<endl;
        return out;
}

ostream& areaData( ostream& out, vector<Material> p, SIZE d, string label, int* grid,
vector<vector<int> >& record)  {

        static int staticCt(0);
        int localCt(0);
        int blank;
        out <<label <<", ";
        vector<Material>::iterator itMat = p.begin();
        while ( itMat != p.end() )      {
                SurfArea sa( SurfArea( d, *itMat ) );
                blank = sa.areaTotal( grid );
                out <<blank  <<", ";
                record[staticCt][localCt] = blank;
                localCt++;
                ++itMat;        }

        itMat = p.begin();
        Material dummy;
        while ( itMat != p.end() )      {
                SurfArea sa( SurfArea( d, *itMat ) );
                dummy = (itMat == --p.end()) ? p.front() : *(itMat + 1);
                blank = sa.areaInterface( grid,  dummy.get_id() );
                out <<blank <<", ";
                record[staticCt][localCt] = blank;
                localCt++;
                ++itMat;        }

        itMat = p.begin();
        while ( itMat != p.end() )      {
                SurfArea sa( SurfArea( d, *itMat ) );
                vector <int> bs = sa.areaBoundary( grid );
                vector <int>::iterator iter = bs.begin();
```

```
                while ( iter != bs.end() )    {
                        out <<*iter  <<", ";
                        record[staticCt][localCt] = *iter;
                        localCt++;
                        ++iter;
                }
                ++itMat;       }
        out <<endl;
        staticCt++;
        return out;
}
ostream& areaStat( ostream& out, vector<vector<int> >& record)       {

        double ave, aveDev, stDev, var, skew, curt;
        vector<double> dummy;
        int maxCol = record.size() * 2 + record.size() * 3;
        int maxRow = record.size();
        out <<"ave, aveDev, stDev, var, skew, curt" <<endl;
        for ( int j = 0; j < maxCol; j++ )    {
                for ( int i = 0; i < maxRow; i++ )
                        dummy.push_back( (double)record[i][j] );
                statData ( dummy, ave, aveDev, stDev, var, skew, curt);
                out <<setprecision(4);
                out <<ave <<", " <<aveDev <<", " <<stDev <<", " <<var <<", " <<skew <<", "
<<curt <<endl;
                dummy.clear(); }

        return out;
}
ostream& areaHeader( ostream& out, vector<Material> p )      {
        out <<"label" <<", ";
        for ( int i = 0; i < p.size(); ++i )
                out << p[i].get_name().substr(0, 3) <<", ";
        for ( int i = 0; i < p.size(); ++i )
                out << p[i].get_name().substr(0, 3) + p[( i == p.size() - 1 ) ? 0 : i +
1].get_name().substr(0, 3) <<", ";
        for ( int i = 0; i < p.size(); ++i )
                out << "R." + p[i].get_name().substr(0, 3) <<", "
                        << "T." + p[i].get_name().substr(0, 3) <<", "
                        << "F." + p[i].get_name().substr(0, 3) <<", ";
        out <<endl;
}
ostream& leastSquare( ostream& out, vector<Material> p, double cl, const SIZE& d,
double*** record )     {

        Energy ni( cl, p[0] ), ysz( cl, p[1] );
        string name = p[2].get_name() + convert( d.D() ) + ".dat";
        Energy comp( name.c_str() );

        int half = d.D() / 2;
        double niTotal[d.D()]; double yszTotal[d.D()]; double coTotal[d.D()];
        double niFirst[half]; double yszFirst[half];
        double coFirst[half];

        cout <<"Char length:  " <<half <<", " <<half <<", " <<half <<endl;

        out <<d <<endl;
        string header = "Num, niTot, ni" + convert( half ) + ", yszTot," +
                                                                                "ysz"
+ convert( half ) + ", coTot, " +
                                                                                "co"
+ convert( half );
        out <<header <<endl;
        double niMax(0), yszMax(0), coMax(0), niMin(1), yszMin(1), coMin(1);
        double niMaxI(0), yszMaxI(0), coMaxI(0), niMinI(1), yszMinI(1), coMinI(1);
        for ( int i = 0; i < d.W(); i++ )     {

                for ( int j = 0; j < d.D(); j++ )     {
                        niTotal[j] = record[i][0][j];
                        yszTotal[j] = record[i][5][j];
                        coTotal[j] = record[i][10][j]; }
```

```
                for ( int j = 0; j < half; j++ )      {
                        niFirst[j] = record[i][0][j];
                        yszFirst[j] = record[i][5][j];
                        coFirst[j] = record[i][10][j]; }
                out <<i <<", ";
                out <<ni.calc(niTotal, d.D()) <<", " <<ni.calc(niFirst, half) <<", ";
                out <<ysz.calc(yszTotal, d.D()) <<", " <<ysz.calc(yszFirst, half) <<", ";
                out <<comp.calc(coTotal, d.D()) <<", " <<comp.calc(coFirst, half) <<endl;

                if ( ni.calc(niTotal, d.D()) > niMax )
                        niMax = ni.get_eng();
                if ( ysz.calc(yszTotal, d.D()) > yszMax )
                        yszMax = ysz.get_eng();
                if ( comp.calc(coTotal, d.D()) > coMax )
                        coMax = comp.get_eng();
                if ( ni.calc(niTotal, d.D()) < niMin )
                        niMin = ni.get_eng();
                if ( ysz.calc(yszTotal, d.D()) < yszMin )
                        yszMin = ysz.get_eng();
                if ( comp.calc(coTotal, d.D()) < coMin )
                        coMin = comp.get_eng();

                if ( ni.calc(niFirst, half) > niMaxI )
                        niMaxI = ni.get_eng();
                if ( ysz.calc(yszFirst, half) > yszMaxI )
                        yszMaxI = ysz.get_eng();
                if ( comp.calc(coFirst, half) > coMaxI )
                        coMaxI = comp.get_eng();
                if ( ni.calc(niFirst, half) < niMinI )
                        niMinI = ni.get_eng();
                if ( ysz.calc(yszFirst, half) < yszMinI )
                        yszMinI = ysz.get_eng();
                if ( comp.calc(coFirst, half) < coMinI )
                        coMinI = comp.get_eng();
        }

        out <<endl <<endl;
        out <<"max, " <<niMax <<", " <<niMaxI <<", " <<yszMax <<", " <<yszMaxI <<", "
<<coMax <<", " <<coMaxI <<endl;
        out <<"min, " <<niMin <<", " <<niMinI <<", " <<yszMin <<", " <<yszMinI <<", "
<<coMin <<", " <<coMinI <<endl;

        out <<"avg" <<endl;
        double ave1(0), ave2(0), ave3(0), a(0);
        vector<double> dummy1(d.W()), dummy2(d.W()), dummy3(d.W());
        for ( int k = 0; k < d.D(); k++ )     {
                int j1(0), j2(5), j3(10);
                for ( int i = 0; i < d.W(); i++ )     {
                        dummy1[i] = record[i][j1][k];
                        dummy2[i] = record[i][j2][k];
                        dummy3[i] = record[i][j3][k];  }
                statData ( dummy1, ave1, a, a, a, a, a);
                statData ( dummy2, ave2, a, a, a, a, a);
                statData ( dummy3, ave3, a, a, a, a, a);
                niTotal[k] = ave1;
                yszTotal[k] = ave2;
                coTotal[k] = ave3;
        }
        for ( int j = 0; j < half; j++ )
                niFirst[j] = niTotal[j];
        for ( int j = 0; j < half; j++ )
                yszFirst[j] = yszTotal[j];
        for ( int j = 0; j < half; j++ )
                coFirst[j] = coTotal[j];

        out <<"avg, ";
        out <<ni.calc(niTotal, d.D()) <<", " <<ni.calc(niFirst, half) <<", ";
        out <<ysz.calc(yszTotal, d.D()) <<", " <<ysz.calc(yszFirst, half) <<", ";
        out <<comp.calc(coTotal, d.D()) <<", " <<comp.calc(coFirst, half) <<endl;

        return out;  }
```

Counter.h

```
#ifndef GUARD_Counter_h
#define GUARD_Counter_h

//      Counter.h
/*      Class members:
                int count
        Member functions:
                Counter a():            default constructor -> 0
                Counter a(value):       constructor to integer value
                inc_count():            increment count
                reset():                        reset count
                get_count():            get count
                bool_count():           bool determine if equal to integer value
        Friend functions:
                Overloaded << for output to screen
*/

#include <iostream>
#include <fstream>
using std::ostream;
using std::cout;
using std::endl;

class Counter
{
private:
        unsigned int count;
public:
        Counter() : count(0) { }
        Counter(int value) : count(value) { }
        void incCount(){ count++; }
        void reset()            { count = 0; }
        int get_count()const    { return count; }
        bool operator == (int value ) { return count == value; }
        bool operator > ( int value ) { return count > value; }
        bool operator < ( int value ) { return count < value; }
        double operator /( Counter c ){ return (double)count / c.get_count(); }
        int operator %( int value ) { return count % value; }
        friend ostream& operator <<(ostream& os, const Counter& c);
};

#endif
```

Counter.cpp

```
#include "Counter.h"
ostream& operator <<(ostream& os, const Counter& c)
{
        os <<c.count;
        return os;
}
```

SIZE.h

```
#ifndef GUARD_SIZE_h
#define GUARD_SIZE_h

#include <cmath>
#include <stdexcept>
using std::domain_error;
#include <iomanip>
using std::setw;
#include <iostream>
#include <fstream>
using std::ostream;
using std::istream;
using std::cout;
using std::endl;

//Constructors Default is 0
//                              (n) sets all equal to n
//                              (w, y, z) sets each equal
//                              each one calculate NUM
```

```
//Destructor
//Accessors
//Manipulators
//      DW, DH, DD      calculates difference between inner and outer
//                              sizes.  Order in functions does not matter
//  min                 finds smallest dimension and outputs
//Operators
//      *                       multiplies by scaler
//      >                       boolean test > relationship for each dimension
//                              will output error
//Output
//      >>                      one line prints all data members
//Friends
//      borderTest      boolean tells if coordinates are on border between
//                              two sizes

class SIZE {
private:
        int WIDTH;
        int HEIGHT;
        int DEPTH;
        int NUM;
public:
        SIZE () : WIDTH(0), HEIGHT(0), DEPTH(0) {
                NUM = WIDTH * HEIGHT * DEPTH;          }
        SIZE (int n) : WIDTH(n), HEIGHT(n), DEPTH(n) {
                NUM = WIDTH * HEIGHT * DEPTH;          }
        SIZE (int w, int h, int d) : WIDTH(w), HEIGHT(h), DEPTH(d) {
                NUM = WIDTH * HEIGHT * DEPTH;          }
        ~SIZE() {       }
        int W() const { return WIDTH; }
        int H() const { return HEIGHT; }
        int D() const { return DEPTH; }
        int S() const { return NUM; }
        int DW(const SIZE& s)  {return (int) fabs(s.W() - WIDTH) / 2; }
        int DH(const SIZE& s)  {return (int) fabs(s.H() - HEIGHT) / 2; }
        int DD(const SIZE& s)  {return (int) fabs(s.D() - DEPTH) / 2; }
        int min();
        SIZE operator *(int n) const;
        SIZE operator +(int n) const;
        bool operator >(SIZE s) const;
        friend ostream& operator <<(ostream& os, const SIZE& s);
        friend istream& operator >>(istream&, SIZE&);
        friend bool borderTest(int, int, int, SIZE s, SIZE p);
};

#endif
```

SIZE.cpp

```
#include "SIZE.h"
//CONSTRUCTORS - header file
//DESTRUCTORS - header file
//ACCESSORS - header file
//MANIPULATORS
int SIZE::min()
{
        int temp = DEPTH;
        if ( WIDTH < DEPTH ) { temp = WIDTH; }
        if ( HEIGHT < temp ) { temp = HEIGHT; }
        return temp;
}
//OVERLOADED OPERATORS
SIZE SIZE:: operator *(int n) const
{
        SIZE temp(n * WIDTH, n * HEIGHT, n * DEPTH);
        return temp;
}
SIZE SIZE:: operator +(int n) const
{
        SIZE temp(n + WIDTH, n + HEIGHT, n + DEPTH);
```

```
        return temp;
}
bool SIZE:: operator >(SIZE s) const
{
        bool a = WIDTH > s.W();
        bool b = HEIGHT > s.H();
        bool c = DEPTH > s.D();
        if ( !((a == b) && (b == c)) )
                throw domain_error("Sizes contradict"  );
        if ( WIDTH > s.W() )
                return true;
        else
                return false;
}
//OUTPUT
ostream& operator <<(ostream& os, const SIZE& s)
{
        os <<"WIDTH = " <<s.WIDTH <<", ";
        os <<"HEIGHT = " <<s.HEIGHT <<", ";
        os <<"DEPTH = " <<s.DEPTH <<", ";
        os <<"NUM = " <<s.NUM <<endl;
        return os;
}
istream& operator >>(istream& in, SIZE& s)    {
        char ch;
        in >> s.WIDTH >>ch >>s.HEIGHT >>ch >>s.DEPTH;
        s.NUM = s.WIDTH * s.HEIGHT * s.DEPTH;
        return in;
}

//FRIENDS
bool borderTest(int w, int h, int d, SIZE s, SIZE p)
{
        bool a = w < s.DW(p);
        bool b = h < s.DH(p);
        bool c = d < s.DD(p);
        bool e = w >= ((s.WIDTH > p.WIDTH) ? s.WIDTH - s.DW(p) : p.WIDTH - s.DW(p));
        bool f = h >= ((s.HEIGHT > p.HEIGHT) ? s.HEIGHT - s.DH(p) : p.HEIGHT - s.DH(p));
        bool g = d >= ((s.DEPTH > p.DEPTH) ? s.DEPTH - s.DD(p) : p.DEPTH - s.DD(p));
        if ( a || b || c || e || f || g )
                        return true;
                else
                        return false;}
```

Coord.h

```
#ifndef GUARD_Coord_h
#define GUARD_Coord_h

#include "SIZE.h"
#include <cmath>
#include <vector>
using std::vector;
#include <iomanip>
using std::setw;
#include <iostream>
#include <fstream>
using std::ostream;
using std::cout;
using std::endl;

//Constructors Default 0
//                              (n) sets all to n
//                              (x, y, z) sets each one
//Destructor
//Accessors             .getvariablename
//Manipulator
// set..                setvariablename each one
//                              set(x,y,z) will set all
//Outputs
// <<                   (x,y,z) to line
```

```
//Friends
//      index          returns 3D index value for given Coord and SIZE
//      anyEqual       boolean if any coordinates are equal to another
//  distance   calculates distance between two coordinates

class Coord    {
private:
        int x;
        int y;
        int z;
public:
        Coord () : x(0), y(0), z(0)    {       }
        Coord (int n) : x(n), y(n), z(n)      {       }
        Coord (int i, int j, int k) : x(i), y(j), z(k)       {      }
        ~Coord()       {       }
        int getx() { return x; }
        int gety() { return y; }
        int getz() { return z; }
        int setx(int);
        int sety(int);
        int setz(int);
        void set(int, int, int);
        bool operator == (Coord& c) { return ( x == c.x && y == c.y && z == c.z ); }
        friend ostream& operator <<(ostream& os, const Coord& c);
        friend int index(const Coord&, const SIZE&);
        friend bool anyEqual(const Coord&, const Coord&);
        friend bool anyNext(const Coord&, const Coord&);
        friend bool anyNext(const Coord&, vector<Coord>);
        friend double distance(const Coord& b, const Coord& c);
};
#endif
```

Coord.cpp

```
#include "Coord.h"
//CONSTRUCTORS - header file
//DESTRUCTOR - header file
//ACCESSORS - header file
//MANIPULATOR FUNCTIONS
int Coord::setx( int value)    {
        return x = value;      }
int Coord::sety( int value) {
        return y = value;      }
int Coord::setz( int value) {
        return z = value;      }
void Coord::set(int a, int b, int c)  {
        x = a; y = b; z = c;
}
//OUTPUT FUNCTIONS
ostream& operator <<(ostream& os, const Coord& c)     {
        os <<"( " <<c.x <<", " <<c.y <<", " <<c.z <<" )";
}
//FRIEND FUNCTIONS
int index(const Coord& c, const SIZE& s)      {
        return c.x * s.H() * s.D() + c.y * s.D() + c.z;
}
bool anyEqual(const Coord& a, const Coord& b) {
        return (a.x == b.x) || (a.y == b.y) || (a.z == b.z);
}
bool anyNext(const Coord& a, const Coord& b)  {
        int e = a.x - b.x; int f = a.y - b.y; int g = a.z - b.z;
        return ( abs( e ) + abs( f ) + abs( g ) <= 1 ) ;
}
bool anyNext(const Coord& a, vector<Coord> b) {
        bool temp = false;
        vector<Coord>::iterator iter = b.begin();
        for (iter = b.begin(); iter != b.end(); ++iter ) {
                if ( anyNext( a, *iter ) == true )
                        temp == true;
                cout <<*iter <<endl;
        }
        return temp;
```

```
}
double distance ( const Coord& b, const Coord& c)     {
        return sqrt(pow(b.x-c.x,2) + pow(b.y-c.y,2) + pow(b.z-c.z,2));
}
```

Material.h

```
#ifndef GUARD_Material_h
#define GUARD_Material_h

#include <iostream>
using std::cerr;
using std::cout;
using std::endl;
using std::cin;

#include <fstream>
using std::ifstream;
using std::istream;
using std::ostream;

#include <string>
using std::string;

#include <stdexcept>
using std::domain_error;

//Constructors Default is "blank", .5, 1
//                              or specify name, vf, and id
//Destructor
//Accessors            get_(variable name)
//Private Func
//  Error              Checks vf is between 0 and 1
//Operators
// +                   adds vf of Materials
// --                  finds complement of vf and increments id
//Output
// >>                  Name, VF, and ID to one line
//Input
// <<                  Input name, VF, and ID


class Material {
private:
        string name;
        double vf;
        int id;
        void error();
public:
        Material ();
        Material (string, double, int);
        ~Material();
        string get_name() const;
        double get_vf() const;
        int get_id() const;
        Material operator +(const Material& old);
        Material operator --();
        //OUTPUT FUNCTION
        friend ostream& operator <<( ostream& out, const Material& m );
        friend istream& operator >>( istream& in, Material& m);
};

#endif
```

Material.cpp

```
#include "Material.h"

// CONSTRUCTOR FUNCTIONS
Material::Material(): name("blank"), vf(.5), id(1)    {
}
Material::Material(string n, double v, int i) : name(n), vf(v), id(i) {
        error();
}
```

```
// DESTRUCTOR FUNCTIONS
Material::~Material()  {
}

// PRIVATE FUNCTIONS
void Material::error()
{
        if (vf > 1 || vf < 0)
                throw domain_error("Volume fractions is not between 1 and 0.");
}

// ACCESSOR FUNCTIONS
string Material::get_name() const { return name; }
double Material::get_vf() const { return vf; }
int Material::get_id() const { return id; }
// OVERLOADED OPERATORS
Material Material::operator +(const Material& old)
{
        Material temp("temp", 1, 1);
        temp.vf = vf + old.vf;
        (id > old.id)?temp.id = id:temp.id = old.id;
        temp.error();
        return temp;
}

Material Material::operator --()
{
        Material temp("Complement", 1, id + 1);
        temp.vf = 1 - vf;
        temp.error();
        return temp;
}
// OUTPUT FUNCTIONS
ostream& operator<<(ostream& out, const Material& m)
{
        out << "Name:  " <<m.name;
        out << ", VF:  " <<m.vf;
        out << ", ID:  " <<m.id;
        return out;
}
//INPUT FUNCTIONS
istream& operator>>(istream& in, Material& m)
{
        in >>m.name >>m.vf >>m.id;
        int ct = 0;
        while ( (m.vf > 1 || m.vf < 0) && ct < 5)     {
                cout <<"Volume fraction must be in decimals." <<endl;
                cin >>m.vf;
                ct++;   }
        m.error();
        return in;}
```

Stochastic1.h

```
#ifndef GUARD_Stochastic1_h
#define GUARD_Stochastic1_h

#include "jmath.h"
#include "SIZE.h"
#include "Coord.h"
#include "Material.h"

#include <iostream>
using std::cerr;
using std::cout;
using std::endl;
using std::cin;

#include <fstream>
using std::ifstream;
using std::ofstream;
```

```
#include <vector>
using std::vector;

class Hits     {
private:
        int*** row;
protected:
        SIZE dim;
        int dir;
        int per;
        int sz;
        int temp;
        int indexTwo;
        int indexThree;
        int (Hits::*pt2indexes)(int);
        virtual bool criteria(int, int, int) = 0;
public:
        void initialize(int, int, int, SIZE );
        ~Hits();
        int get(int, int, int);
        int indexRow(int);
        int indexCol(int);
        int indexDep(int);
        virtual void updateIndex(int, int, int);
        virtual int accept(int);
        virtual int accept(int, int, int, int);
        virtual int add(int*, int, int ) = 0;
        virtual int setMat( Material& ) = 0;
        friend ostream& operator <<( ostream& out, const Hits& h );
};

class TwoPoint : public Hits {
protected:
        bool criteria(int, int, int);
public:
        int add(int*, int, int);
        int setMat( Material& );
};
class LinealChord : public TwoPoint {
public:
        int add(int*, int, int);
};
class MixedPhase : public TwoPoint {
        int n;
public:
        void initialize(int r, int p, int o, SIZE d ) {
                Hits::initialize( r, p, o, d );
                int i = 0; n = i;      }
        int setMat( Material& mat );
        bool criteria( int, int, int );
};
class ClusterFunc : public Hits {
        int current;
        vector<Coord> all;
        int* coordMap;
        Coord (ClusterFunc::*pt2coords)(int);
public:
        ClusterFunc() : current(0) {   }
        void initialize( int r, int p, int o, SIZE d );
        Coord coordRow( int );
        Coord coordCol( int );
        Coord coordDep( int );
        int setMat( Material& ) { return 0; }
        void fillAll( int*, int mat );
        void updateIndex(int, int, int );
        bool criteria(int, int, int );
        bool criteria(Coord&, Coord&);
        int add(int* arr, int r, int mat);
};

class Energy   {
```

```
private:
        double eng;
        double r_func[400];
        double charLength;
public:
        Energy();
        Energy( double, const Material );
        Energy( double, const Material, char );
        Energy( double, double, const Material, char );
        Energy( long double, long double, long double, long double, long double, long
double, long double, double, long double );
        Energy( const char* );
        ~Energy();
        double get_eng() const;
        double get_cl() const;
        double calc( const vector<double> );
        double calc( const double*, int );
        friend ostream& operator <<( ostream&, const Energy& );
};

template<class H>
class Stochastic  {
private:
        int radius;
        int per;
        Material mat;
        SIZE dim;
        int* trials;
        double* func;          //function is always current even to temp changes
        double* o_func;
        vector<int> hist;
        void calcRadius();
        void initialize();
protected:
        H width, height, depth;
public:
        Stochastic();
        Stochastic(Material m, SIZE d, int radius = 25, int per = 1);
        Stochastic(Material m, Material n, SIZE d, int radius = 25, int per = 1);
        Stochastic(int id, double vf, int X, int Y, int Z);
        ~Stochastic();
        int get_radius();
        double get_func(int r = 0);
        vector<double> funcOut();
        void total(int*);
        void temp(int*, Coord&, Coord& );
        void accept( Coord&, Coord& );
        void reject();
        void statusHits(int, Coord&, Coord&);
        friend ostream& operator <<( ostream& out, const Stochastic<H>& s )  {
                out <<s.mat <<s.dim;
                out << "Radius:  " <<s.radius <<endl;
        return out;            }
};
//
//STOCHASTIC CLASS
//
//CONSTRUCTOR FUNCTIONS
template <class H>
Stochastic<H>::Stochastic() : radius(25), per(0), mat("DEFAULT", .5, 1), dim(30)
{
        initialize();
}
template <class H>
Stochastic<H>::Stochastic(Material m, SIZE d, int r, int p ) : mat(m), dim(d)
{
        radius = r;
        per = p;
        initialize();
}
template <class H>
```

```
Stochastic<H>::Stochastic(Material m, Material n, SIZE d, int r, int p ) : mat(m), dim(d)
        radius = r;
        per = p;
        initialize();
        width.setMat( n );
        height.setMat( n );
        depth.setMat( n );
}
template <class H>
Stochastic<H>::Stochastic(int id, double vf, int X, int Y, int Z)
{
        Material blank("BLANK", vf, id);
        SIZE blank1(X, Y, Z);
        mat = blank;   dim = blank1;
        radius = X;
        per = (int)(radius / 2);
        initialize();
}
//PRIVATE FUNCTIONS
template <class H>
void Stochastic<H>::calcRadius()      {
        if ( radius > dim.min() && dim.min() >= 10 )  {
                radius = dim.min() - 5;
                cout <<"The radius has been changed to " <<radius <<endl;    }
        else if ( radius > dim.min() ) {
                radius = 5;
                cout <<"The radius has been changed to " <<radius <<endl;    }
}
template <class H>
void Stochastic<H>::initialize()
{
        calcRadius();
        trials = new int[radius];
        func = new double[radius];
        o_func = new double[radius];
        for ( int i = 0; i < radius; i++ ) {
                func[i] = 0; o_func[i] = 0;
                trials[i] = (dim.W()- i) * dim.H() * dim.D()
                             + dim.W() * (dim.H() - i) * dim.D()
                                 + dim.W() * dim.H() * (dim.D() - i);
        }
        for ( int i = 0; i < per; i++ )
                trials[i] = dim.S() + dim.S() + dim.S();
        width.initialize(radius, per, 1, dim);
        height.initialize(radius, per, 2, dim);
        depth.initialize(radius, per, 3, dim);
}
template <class H>
Stochastic<H>::~Stochastic() { }
//ACCESSOR FUNCTIONS
template <class H>
int Stochastic<H>::get_radius()       {
        return radius; }
template <class H>
double Stochastic<H>::get_func(int r) {
        return func[r];        }
template <class H>
vector<double> Stochastic<H>::funcOut()       {
        vector<double> temp;
        for ( int i = 0; i < radius; i++ )
                temp.push_back(func[ i ]);
        return temp;   }
//MANIPULATOR FUNCTIONS
template <class H>
void Stochastic<H>::total(int* arr)
{
        for ( int r = 0; r < radius; r++ )    {
                int total_Hits = 0;
                for ( int col = 0; col < dim.H(); col++ )     {
                        for ( int dep = 0; dep < dim.D(); dep++ )     {
                                width.updateIndex(col, dep, 0);
```

```
                                width.add(arr, r, mat.get_id());
                                total_Hits = total_Hits + width.accept(r);    }
                        }
                for ( int row = 0; row < dim.W(); row++ )     {
                        for ( int dep = 0; dep < dim.D(); dep++ )     {
                                height.updateIndex(row, dep, 0);
                                height.add(arr, r, mat.get_id());
                                total_Hits = total_Hits + height.accept(r);   }
                        }
                for ( int row = 0; row < dim.W(); row++ )     {
                        for ( int col = 0; col < dim.H(); col++ )     {
                                depth.updateIndex(row, col, 0);
                                depth.add(arr, r, mat.get_id());
                                total_Hits = total_Hits + depth.accept(r);    }
                        }
                //cout <<"Totat hits: " <<total_Hits <<" at r " <<r <<endl;
                func[r] = (double)total_Hits / trials[r];
        }
}
template <class H>
void Stochastic<H>::temp(int* arr, Coord& a, Coord& b)
{
        hist.clear();
        int old, new_;

        int r1 = a.getx();     int c1 = a.gety();     int d1 = a.getz();
        int r2 = b.getx();     int c2 = b.gety();     int d2 = b.getz();

        for ( int r = 0; r < radius; r++ )    {

                old = width.get(r, c1, d1) + width.get(r, c2, d2) +
                          height.get(r, r1, d1) + height.get(r, r2, d2) +
                          depth.get(r, r1, c1) + depth.get(r, r2, c2);

                width.updateIndex(c1, d1, r1);
                hist.push_back(width.add(arr, r, mat.get_id()));
                width.updateIndex(c2, d2, r2);
                hist.push_back(width.add(arr, r, mat.get_id()));
                height.updateIndex(r1, d1, c1);
                hist.push_back(height.add(arr, r, mat.get_id()));
                height.updateIndex(r2, d2, c2);
                hist.push_back(height.add(arr, r, mat.get_id()));
                depth.updateIndex(r1, c1, d1);
                hist.push_back(depth.add(arr, r, mat.get_id()));
                depth.updateIndex(r2, c2, d2);
                hist.push_back(depth.add(arr, r, mat.get_id()));

                new_ = hist[0+6*r] + hist[1+r*6] + hist[2+r*6]
                         + hist[3+r*6] + hist[4+r*6] + hist[5+r*6];
                o_func[r] = func[r];
                func[r] = ( func[r] * trials [r] + new_ - old ) / trials[r];
                //if ( r == 0 )
                //cout <<width <<endl;
        }
}
template <class H>
void Stochastic<H>::accept(Coord& a, Coord& b)
{
        int r1 = a.getx();     int c1 = a.gety();     int d1 = a.getz();
        int r2 = b.getx();     int c2 = b.gety();     int d2 = b.getz();
        for ( int r = 0; r < radius; r++ )    {
                width.accept(r, c1, d1, hist[0+6*r]);
                width.accept(r, c2, d2, hist[1+6*r]);
                height.accept(r, r1, d1, hist[2+6*r]);
                height.accept(r, r2, d2, hist[3+6*r]);
                depth.accept(r, r1, c1, hist[4+6*r]);
                depth.accept(r, r2, c2, hist[5+6*r]);
        }
        hist.clear();
}
template <class H>
```

```
void Stochastic<H>::reject()   {
        for ( int r = 0; r < radius; r++ )
                func[r] = o_func[r];
}
template <class H>
void Stochastic<H>::statusHits(int r, Coord& a, Coord& b )   {
        cout <<"Width: " <<width;
        cout <<"Height: " <<height;
        cout <<"Depth:  " <<depth;
}
#endif
```

Stochastic1.cpp

```
#include "Stochastic1.h"
//
//HITS CLASS
//
//CONSTRUCTOR FUNCTIONS
void Hits::initialize(int r, int p, int o, SIZE d )
{
        dim = d;
        dir = o;
        per = p;
        switch ( dir ) {
                case 1 :
                        pt2indexes = &Hits::indexRow;
                        sz = dim.W();
                        indexTwo = dim.H();    indexThree = dim.D();
                        break;
                case 2 :
                        pt2indexes = &Hits::indexCol;
                        sz = dim.H();
                        indexTwo = dim.W();    indexThree = dim.D();
                        break;
                case 3 :
                        pt2indexes = &Hits::indexDep;
                        sz = dim.D();
                        indexTwo = dim.W();    indexThree = dim.H();
                        break;
                default:
                        pt2indexes = &Hits::indexRow;
                        sz = dim.W();
                        indexTwo = dim.H();    indexThree = dim.D();
                        break;
        }
        row = new int**[r];
        for ( int i = 0; i < r; i++ )
                *(row + i) = new int*[indexTwo];
        for ( int i = 0; i < r; i++ )  {
                for ( int j = 0; j < indexTwo ; j++ )
                        *(*(row + i) + j) = new int[indexThree];      }
        for ( int i = 0; i < r; i++ )  {
                for ( int j = 0; j < indexTwo; j++ )          {
                        for ( int k = 0; k < indexThree; k++ )        {
                                row[i][j][k] = 0;
}}}
}
//DESTRUCTOR FUNCTION
Hits::~Hits()  {}
//ACCESSOR FUNCTIONS
int Hits::get(int r, int i2, int i3)  {
        return row[r][i2][i3]; }
//MANIPULATOR FUNCTION
int Hits::indexRow(int row_)   {
        return row_ * dim.H() * dim.D() + indexTwo * dim.D() + indexThree;
}
int Hits::indexCol(int col) {
        return indexTwo * dim.H() * dim.D() + col * dim.D() + indexThree;
}
int Hits::indexDep(int dep)    {
        return indexTwo * dim.H() * dim.D() + indexThree * dim.D() + dep;
```

```
}
void Hits::updateIndex(int i_2, int i_3, int) {
	indexTwo = i_2;
	indexThree = i_3;
}
int Hits::accept(int r)	{
	row[r][indexTwo][indexThree] = temp;
	return temp;
}
int Hits::accept(int r, int i2, int i3, int val)	{
	row[r][i2][i3] = val;
	return val;
}
//FRIENDS
ostream& operator <<( ostream& out, const Hits& h)
{
		/*for ( int i = 0; i < 1; i++ )	{
		out <<endl;
		for ( int j = 0; j < 10; j++ )	{
			out <<endl;
			for ( int k = 0; k < 10; k++ )	{
				out <<h.row[i][j][k] <<"  ";
			}}}
		out <<endl;*/
		out <<"Index 2:  " <<h.indexTwo  <<", ";
		out <<"Index 3:  " <<h.indexThree  <<", ";
		out <<"Temp Hits:  " <<h.temp <<endl;
		return out;
}

//TWO POINT CORRELATION FUNCTION
bool TwoPoint::criteria(int id1, int id2, int mat )	{
	return ( id1 == mat && id2 == mat );
}
int TwoPoint::add(int* arr, int r, int mat)	{
	temp = 0;
	int ct = 0; int spacer = ct + r;
	int space_1, space_2;

	int totCt = 0;

	while ( totCt < sz /*spacer < sz*/ )
	{
		//cout <<"In loop  " <<ct <<endl;
		//Periodic
		if ( r < per && spacer >= sz )
			spacer = spacer - sz;
		//End Periodic
		space_1 = arr[(*this.*pt2indexes)(ct)];
		space_2 = arr[(*this.*pt2indexes)(spacer)];
		if (criteria (space_1, space_2, mat) == true )
			temp++;
		//if ( r == 1 )
		//cout <<"1:  " <<ct <<"  2: " <<spacer <<" : " <<criteria (space_1,
space_2, mat) <<endl;
		ct++;
		spacer = ct + r;
		if ( r < per )
			totCt = ct;
		else
			totCt = spacer;	}
	return temp;
}
int TwoPoint::setMat( Material& c )	{
	return 0;
}
//LINEAL CHORD FUNCTION
int LinealChord::add(int* arr, int r, int mat)		{

	temp = 0;
	int ct = 0;
```

```
        int spacer = ct + r;
        int center = ct + 1;
        int center1 = ct + 1;
        bool query = false;
        int space_1, space_2;

        int totCt = 0;

        while ( totCt < sz ) {//( spacer < sz )       {
                if ( r < per && spacer >= sz )
                        spacer = spacer - sz;
                space_1 = arr[(*this.*pt2indexes)(ct)];
                space_2 = arr[(*this.*pt2indexes)(spacer)];
                if (criteria(space_1, space_2, mat) == true ) {
                        center = ct + 1;
                        center1 = ct + 1;
                        if ( r < per && spacer >= sz ) {
                                center = ct + 1 - sz;
                                center1 = ct + 1 - sz; }
                        //center1 = ct + 1;
                        while ( center1 < spacer )     {
                                if ( arr[(*this.*pt2indexes)(center1)] != mat )      {
                                        //added center1 above
                                        query = true; break;   }
                                center1++;                     }
                        if ( query == false ) temp++;          }
                ct++;
                spacer = ct + r;
                query = false;
                if ( r < per )
                        totCt = ct;
                else
                        totCt = spacer;
        }
        return temp;
}

//MIXED PHASE FUNCTION
bool MixedPhase::criteria(int id1, int id2, int mat ){
        return ( id1 == mat && id2 == n ); // || ( id1 == n && id2 == mat )
                //      || ( id1 == mat && id2 == mat ) || ( id1 == n && id2 == n ) ;
}
int MixedPhase::setMat( Material& mat )       {
        n = mat.get_id();
        return 0;      }

//CLUSTER FUNCTION
void ClusterFunc::initialize( int r, int p, int o, SIZE d )  {
        Hits::initialize( r, p, o, d );
        switch ( dir ) {
                case 1 :
                        pt2coords = &ClusterFunc::coordRow;
                break;
                case 2 :
                        pt2coords = &ClusterFunc::coordCol;
                        break;
                case 3 :
                        pt2coords = &ClusterFunc::coordDep;
                        break;
                default:
                        pt2coords = &ClusterFunc::coordRow;
                        break;  }
        coordMap = new int[ dim.S() ];
        for ( int i = 0; i < dim.S(); i++ )
                coordMap[i] = 0;
}
Coord ClusterFunc::coordRow( int row_ )       {
        Coord dummy(row_, indexTwo, indexThree);
        return dummy;  }
Coord ClusterFunc::coordCol( int col ){
        Coord dummy(indexTwo, col, indexThree);
```

```
        return dummy;  }
Coord ClusterFunc::coordDep( int dep ){
        Coord dummy(indexTwo, indexThree, dep);
        return dummy;  }
void ClusterFunc::fillAll(int* arr, int mat)  {

        Coord temp;
        for ( int i = 0; i < dim.W(); i++ )   {
                for ( int j = 0; j < dim.H(); j++ )   {
                        for ( int k = 0; k < dim.D(); k++ )   {
                        temp.set(i, j, k);
                        if ( arr[ index( temp , dim) ] == mat )
                                all.push_back( temp );
        }}}

        int ct = 1, c = 0;     //counter of clusters
        vector<Coord> cluster;
        vector<Coord> dummy = all;
        vector<Coord>::iterator itDummy = dummy.begin();
        bool status = false;

        while ( !dummy.empty() )       {
                cluster.push_back( *(dummy.begin()) );
                c = 0;
                while ( c < cluster.size() )   {
                        itDummy = dummy.begin();
                        temp = *(cluster.begin() + c);
                        while ( itDummy != dummy.end() )      {
                                if ( anyNext(temp, *itDummy) == true ){
                                        cluster.push_back(*itDummy);
                                        coordMap[ index( *itDummy, dim ) ] = ct;
                                        itDummy = dummy.erase(itDummy);
                                        status = true;
                                }
                                else
                                        itDummy++;     }
                        c++;    }
                if ( status == false )
                        coordMap[ index( *(itDummy - 1), dim) ] = ct;
                status = false;
                ct++;
                cluster.clear();
        }
}
void ClusterFunc::updateIndex(int i_2, int i_3, int d)       {
        indexTwo = i_2;
        indexThree = i_3;
        current = d;
}
bool ClusterFunc::criteria(int id1, int id2, int mat){
        return ( id1 == mat && id2 == mat );
}
bool ClusterFunc::criteria(Coord& a, Coord& b)       {
        return ( coordMap[index(a, dim)] == coordMap[index(b, dim)] );       }
int ClusterFunc::add(int* arr, int r, int mat)               {

        if ( all.size() == 0 )
                fillAll( arr, mat );

        temp = 0;
        int ct = 0; int spacer = ct + r;
        Coord pt_1, pt_2;
        int space_1, space_2;
        while ( spacer < sz )
        {
                pt_1 = (*this.*pt2coords)(ct);
                pt_2 = (*this.*pt2coords)(spacer);
                space_1 = arr[(*this.*pt2indexes)(ct)];
                space_2 = arr[(*this.*pt2indexes)(spacer)];
                if (criteria (space_1, space_2, mat ) == true
                                && criteria (pt_1, pt_2 ) == true )
```

```
                        temp++;
                ct++;
                spacer = ct + r;
        }
        return temp;
}

//ENERGY FUNCTIONS
//CONSTRUCTOR FUNCTIONS
Energy::Energy() : eng( 0 )    { }
Energy::Energy(double dia, const Material m ) : eng( 0 )     {
        charLength = dia;
        for ( int i = 0; i <= dia; i++ )
                r_func[i] = overlapping_sphere(dia, i, m.get_vf());
        for ( int i = (int)dia + 1; i < 400; i++ )
                r_func[i] = m.get_vf() * m.get_vf();
}
Energy::Energy(double dia, const Material m, char debye ) : eng( 0 ) {
        charLength = dia;
        for ( int i = 0; i < 400; i++ )       {
                r_func[i] = debye_decay(charLength, i, m.get_vf());
                //if ( i < 20 )
                //      cout <<r_func[i] <<"\t" <<charLength <<"\t" <<m.get_vf() <<endl;
        }
}
Energy::Energy(double dia, double osc, const Material m, char debye ) : eng( 0 ) {
        charLength = dia;
        for ( int i = 0; i < 400; i++ ) {
                r_func[i] = debye_oscill(charLength, osc, i, m.get_vf());
        }
}
Energy::Energy(long double c1, long double c2, long double c3, long double c4, long
double c5, long double c6, long double c7, double length, long double conv ) : eng( 0 )
                for ( int i = 0; i < 400; i++ )       {
                        if ( i*conv > (int)length )
                                r_func[i] = 0;
                        else
                                r_func[i] = poly(c1, c2, c3, c4, c5, c6, c7, conv * i );
                        //if ( i < 100 )
                                /*cout <<i <<"\t" <<conv*i <<"\t" <<r_func[i] <<endl;*/
}
Energy::Energy(const char* name ) : eng(0)    {
        ifstream indata;       indata.open(name);
        if(!indata)    {
                cerr << "Error:  file could not be opened" <<endl;
                exit(1);       }
        int i = 0;
        while ( !indata.eof() )       {
                indata >> * ( r_func + i );
                i++;    }
        indata.close();

        double prev = 0;
        double delta = 0;
        i = 0;
        while ( i < 50 )      {
                delta = fabs( prev - r_func[i] ) * 100 / r_func[i] ;
                if (  delta <= 2 )     {
                        charLength = i;
                        break;  }
                prev = * ( r_func + i );
                i++;    }
}
//DESTRUCTOR FUNCTION
Energy::~Energy()      {}
//ACCESSOR FUNCTIONS
double Energy::get_eng() const { return eng; }
double Energy::get_cl() const { return charLength; }
//MANIPULATOR
double Energy::calc( const vector<double> func )     {
        double e = 0;
```

```
        for ( int r = 0; r < func.size(); r++ )
                e = e + pow(func[r] - r_func[r],2);
        eng = e;
        return eng;
}
double Energy::calc( const double* arr, int rad )     {
        double e = 0;
        for ( int r = 0; r < rad; r++ )
                e = e + pow(arr[r] - r_func[r],2);
        eng = e;
        return eng;
}
ostream& operator <<(ostream& out, const Energy& e)
{
        for ( int i = 0; i < 100; i++ )
                out <<e.r_func[ i ] <<endl;
        return out;
}
```

IdLoc.h

```
#ifndef GUARD_IdLoc_h
#define GUARD_IdLoc_h

#include "random.h"
#include "SIZE.h"
#include "Coord.h"

#include <vector>
using std::vector;

#include <iostream>
using std::endl;
using std::cout;
using std::cin;

//Constructors         default 2 phases
//                                      input number phases and dimensions
//                                      sets up both sequences for rotating numbers
//Destructor
//Accessors                   get_variablename
//Private function
//      findLowerLimits finds indices of 3 bottom arrays
//      findUpperLimits finds indices of 3 top arrays
//Manipulator functions
//      rotate                shifts sequence numbers
//      findId                finds first index in array and then sets internal ids
//  findId                    finds index that does not match given input value
//      findNextId            finds index that will not overlap on given IdLoc
//      setId                 sets index and ids for the current location
//      switchId              changes value in given array to id2
//  resetId                   restores value in given array to id1
//Boolean Functions
//      onInterface           true if on interface
//      onBorder              true if on border
//OUTPUT
//      <<                              Outputs location and id switch
//FRIENDS
//      findOnInterface finds a two indexes on interface using findID and
//                                      findNextId


class IdLoc    {
private:
        SIZE out;
        SIZE inn;
        vector<int> seq1;
        vector<int> seq2;
        Coord loc;
        int num;
        int id1;
        int id2;
```

```
        Coord findLowerLimits();
        Coord findUpperLimits();
public:
        IdLoc();
        IdLoc(int, const SIZE&, const SIZE&);
        ~IdLoc();
        int get_id1() const { return id1; }
        int get_id2() const { return id2; }
        int get_current( int* arr ) const { return arr[ index(loc,out) ]; }
        Coord get_loc() const { return loc; }
        void rotate();
        void findId(int*);
        void findId(int*, int);
        void findNextId(int*, const IdLoc&);
        void setId(int*, int);
        void switchId(int*);
        void resetId(int*);
        bool onInterface(int*);
        bool onBorder();
        friend void findOnInterface(int* arr, IdLoc&, IdLoc&);
        friend ostream& operator <<(ostream& os, const IdLoc& id);
};

#endif
```

IdLoc.cpp

```
#include "IdLoc.h"
//CONSTRUCTOR FUNCTIONS
IdLoc::IdLoc() :
        num(2), out(100), inn(100), loc(0), id1(0), id2(0) {
        seq1.push_back( 1 ); seq1.push_back( 2 );
        seq2.push_back( 2 ); seq2.push_back( 1 );
}

IdLoc::IdLoc(int p, const SIZE& o, const SIZE& i)
: num(p), out(o), inn(i), loc(0), id1(0), id2(0)     {
        seq1.push_back( num );         //Set [0] to max id
        seq2.push_back( num - 1);      //Set [0] to max id - 1

        for (int i = 1; i < num; i++)  {
                seq1.push_back(i);     //pattern should be 3, 1, 2
                seq2.push_back(seq1[i - 1]);  //pattern should be 2, 3, 1
        }
}
//DESTRUCTOR FUNCTION
IdLoc::~IdLoc(){ }

//PRIVATE FUNCTIONS
Coord IdLoc::findLowerLimits() {
        Coord temp(0, 0, 0);
        if ( loc.getx() != 0 )
                        temp.setx(loc.getx() - 1);
        if ( loc.gety() != 0 )
                temp.sety(loc.gety() - 1);
        if ( loc.getz() != 0 )
                temp.setz(loc.getz() - 1);
        return temp;
}
Coord IdLoc::findUpperLimits() {
        Coord temp(loc.getx()+2, loc.gety()+2, loc.getz()+2);
        if ( loc.getx() >= out.W() - 1 )
                        temp.setx(out.W());
        if ( loc.gety() >= out.H() - 1 )
                        temp.sety(out.H());
        if ( loc.getz() >= out.D() - 1 )
                        temp.setz(out.D());
        return temp;
}

//ACCESSOR FUNCTIONS IN HEADER
//MANIPULATOR FUNCTIONS
```

```
void IdLoc::rotate()   {
       int temp;
       for (int i = 0; i < num; i++)  {
              temp = seq1[i];
              seq1[i] = seq2[i];
              seq2[i] = temp;}
}
void IdLoc::findId( int* arr ) {
       loc.set( random(out.W()), random(out.H()), random(out.D()));
       id1 = arr[ index(loc, out) ];
       int i = 0;
       while ( id1 != seq1[i] )
              i++;
       id2 = seq2[i];
}
void IdLoc::findId( int* arr, int id )
{
       loc.set( random(out.W()), random(out.H()), random(out.D()));
       id1 = arr[index(loc, out)];
       while ( id1 == id )    {
              loc.set( random(out.W()), random(out.H()), random(out.D()));
              id1 = arr[index(loc, out)];
       }
       int i = 0;
       while ( id1 != seq1[i] )
              i++;
       id2 = seq2[i];}
void IdLoc::findNextId(int* arr, const IdLoc& id)
{
       char check = 'N';
       int ct = 0;
       while ( check == 'N' ) {
              ct++;
              loc.set( random(out.W()), random(out.H()), random(out.D()));
              while ( anyEqual(loc, id.get_loc()) )
                     loc.set( random(out.W()), random(out.H()), random(out.D()));

              if ( arr[loc.getx()*out.H()*out.D()+loc.gety()*out.D()+loc.getz()] ==
id.get_id2() ) {
                     id1 = id.get_id2();
                     id2 = id.get_id1();
                     check = 'Y';   }
              if ( ct++ > 1000000000 )      {
                     throw domain_error("Phases incorrect");
                     break;  }}}

void IdLoc::setId( int* arr, int id ) {
       id1 = arr[index(loc,out)];
       id2 = id;
       arr[index(loc,out)] = id2;     }
void IdLoc::switchId(int* arr) {
       arr[index(loc,out)] = id2;     }
void IdLoc::resetId(int* arr)  {
       arr[index(loc, out)] = id1;    }
//BOOLEAN FUNCTIONS
bool IdLoc::onInterface(int* arr)     {

       Coord lower = findLowerLimits();
       Coord upper = findUpperLimits();
       Coord current(0, 0, 0);
       int ct=0, tot=0;

       for ( int i = lower.getx(); i < upper.getx(); i++ )   {
              for ( int j = lower.gety(); j < upper.gety(); j++ )   {
                     for ( int k = lower.getz(); k < upper.getz(); k++ )   {
                            current.set(i, j, k);
                            if (arr[ index(current, out) ] == arr[ index(loc, out) ] )
                            ct++;
                     tot++;}}}
       return ( ct != tot );}
bool IdLoc::onBorder() {
```

```
        return
        loc.getx() < out.DW(inn) || loc.getx() >= out.W()-out.DW(inn) ||
        loc.gety() < out.DH(inn) || loc.gety() >= out.H()-out.DH(inn) ||
        loc.getz() < out.DD(inn) || loc.getz() >= out.D()-out.DD(inn);}
//OUTPUT FUNCTIONS
ostream& operator <<(ostream& os, const IdLoc& id)
{
        /*os <<"Current Material conversion:  "  <<endl;
        for (int i = 0; i < id.num; i++)      {
                os <<"Mat " <<id.seq1[i] <<"->" <<id.seq2[i] <<endl;
        }*/
        os <<"Current location:  " <<id.loc;
        os <<" for Material " <<id.id1;
        os <<" -> " <<id.id2 <<endl;
        return os;}
//FRIENDS
void findOnInterface(int* arr, IdLoc& a, IdLoc& b)
{
                a.findId(arr);          //Select first Id
                while (a.onInterface(arr) == 0)       //Check interface
                        a.findId(arr);
                b.findNextId(arr, a);
                int c = 0;
                while (b.onInterface(arr) == 0 )      {
                        b.findNextId(arr, a);
                        c++;
                        if (c > 1000000 ) break;      }
                a.switchId(arr);
                b.switchId(arr);  }
```

VolStatus.h

```
#ifndef GUARD_VolStatus_h
#define GUARD_VolStatus_h
#include "IdLoc.h"
#include <iostream>
using std::cerr;
using std::cout;
using std::endl;
using std::cin;

//Data members
//                              id1, id2 are current materials of IdLoc
//                              border_stat_1,2 states if id1 or 2 are on border
//                              boolean (true if volume fractions changed between
//                                      border and center)
//Constructors booleans default to true
//Destructors
//      reset           sets status back to true
//      check           sets status to true if both a and b are in center
//                                      or on the border.  Also sets id1 and id2
//Output                outputs volume status
//Friend
//      correctVolStatus
//                              finds new index on border and center and
//                              change accordingly

class VolStatus
{
private:
        int id1;
        int id2;
        bool border_stat_1;
        bool border_stat_2;
        bool status;
public:
        VolStatus();
        ~VolStatus();
        bool get() const { return status; }
        void reset();
        bool check(IdLoc&, IdLoc&);
        friend ostream& operator <<(ostream& os, const VolStatus& );
```

```
        friend void correctVolStatus(int*, IdLoc&, IdLoc&, VolStatus& ); };

#endif
```

VolStatus.cpp

```
#include "VolStatus.h"
//CONSTRUCTORS
VolStatus::VolStatus()
        : status(1), border_stat_1(1), border_stat_2(1), id1(0), id2(0)      {       }
//DESTRUCTORS
VolStatus::~VolStatus()        {       }
//ACCESSORS
//MANIPULATORS
void VolStatus::reset()        {
        status = true;}
//BOOLEANS
bool VolStatus::check(IdLoc& a, IdLoc& b)
{
        if ( a.onBorder() == b.onBorder() )
                status = true;
        else
                status = false;

        id1 = a.get_id2();     id2 = b.get_id2(); //Sets to previous Material
        border_stat_1 = a.onBorder();
        border_stat_2 = b.onBorder();
        return status;}
//OUTPUT
ostream& operator <<(ostream& os, const VolStatus& v  )      {
        os <<"ID 1:  " <<v.id1 <<" and Border Status:  " <<v.border_stat_1;
        os <<"  (1 - Border, 0 - Center ) " <<endl;
        os <<"ID 2:  " <<v.id2 <<" and Border Status:  " <<v.border_stat_2 <<endl;
        os <<"Volume Status:  " <<v.status;
        os <<"  (1 - Status Good, 0 - Volume Changed) " <<endl;
        return os;}
//FRIEND FUNCTIONS
void correctVolStatus(int* arr, IdLoc& a, IdLoc& b, VolStatus& v)
{
        a.findId(arr); b.findId(arr);
        while ( a.onBorder() != v.border_stat_1
                || a.get_id1() != v.id1 )
                        a.findId(arr);

        while ( b.onBorder() != v.border_stat_2
                    || b.get_current( arr ) != v.id2
                        || anyEqual(a.get_loc(), b.get_loc()) == true )      {
                        b.findId(arr);}

        a.setId(arr, v.id2 );
        b.setId(arr, v.id1 );   }
```

Schedule.h

```
#ifndef GUARD_Schedule_h
#define GUARD_Schedule_h

//Data members
//      probability            probability of acceptence
//      threshold              determines rate of acceptence
//Constructors         threshold default .0001
//Destructor
//Manipulators
//      deluge                 deluge acceptence algorithm

class Schedule
{
private:
        double probability;
        double threshold;
public:
        Schedule(double threshold = .0001);
        ~Schedule();
        double deluge(double, double);  };
```

```
#endif
```

Schedule.cpp

```
#include "Schedule.h"
//CONSTRUCTORS
Schedule::Schedule(double n) : probability(0) {
        threshold = n;}
//DESTRUCTORS
Schedule::~Schedule()  {}
//MANIPULATORS
double Schedule::deluge(double old, double new_)      {

        if ( new_ < threshold * old + old )
                probability = 1;
        else
                probability = 0;

        return probability;  }
```

InputPar.h

```
#ifndef GUARD_InputPar_h
#define GUARD_InputPar_h
#include "SIZE.h"
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
#include <fstream>
using std::ostream;
using std::istream;
#include <string>
using std::string;

string convert( int );
void queryDefaults( SIZE&, int& );
string queryString();

class InputPar {
public:
        bool queryStatus();
        SIZE inputSize();
        int inputNum();  };
class Label    {
        bool type;
        char charLabel;
        int intLabel;
        int ct;
public:
        Label();
        void setType();
        void incLabel();
        string getLabel();
};
class FileCall {
private:
        string dir;
        string id;
        string des;
        string inc;
        string ext;
public:
        FileCall();
        FileCall(string, string, string, string inc = "", string ext = ".dat" );
        void updateId( string n)       { id = n;       }
        void updateInc( string n )     { inc = n;      }
        string getDir(){ return dir; }
        string getId() { return id;  }
        string getDes() { return des; }
        string getInc() { return inc; }
        string getExt() { return ext; }
        string name();};
```

```
#endif
```

InputPar.cpp

```
#include "InputPar.h"

string convert( int num )      {

        string temp("0");
        int dig(1), factor(10);
        while ( num % factor != num )  {
                dig++;
                factor = 10 * factor;
                temp.push_back('0');           }

        int var = num;
        char x;
        for ( int i = 0; i < dig ; i ++ )      {
                x = var % 10 + 48;
                temp[dig - 1 - i] = x;
                var = ( var - var % 10 ) / 10;}
        return temp;}

bool InputPar::queryStatus()   {
        char inp;
        cout <<"Use default parameters Y/N" <<endl;
        cin >>inp;
        if ( inp == 'Y' || inp == 'y' )
                return false;
        else
                return true;   }

SIZE InputPar::inputSize()     {
        int num;
        cout <<"Input size as integer" <<endl;
        cin >>num;
        SIZE temp(num);
        return temp;   }

int InputPar::inputNum()       {
        int num;
        cout <<"Input desired number of realizations" <<endl;
        cin >>num;
        return num;    }

Label::Label(): type(false), charLabel('a'), intLabel(1), ct(0)      {       }
void Label::setType()  {
        char inp;
        cout <<"Input Y/y for seq or N/n for char:  " <<endl;
                cin >> inp;
        if ( inp == 'Y' || inp == 'y' )
                type = false;
        else
                type = true;}
void Label::incLabel() {
        charLabel++;
        intLabel++;
        if ( ct == 26 )
                charLabel = 'a';
        ct++;   }
string Label::getLabel()       {
        string temp("blank");
        if ( type == true )            {
                temp = charLabel;
        if ( ct > 26 )
                temp = temp + charLabel;       }
        else if ( type == false )
                temp = "seq" + convert( intLabel );

        return temp;   }
FileCall::FileCall() : dir(""), id("cfiles"), des(""), inc(""), ext(".dat")  {
}
```

```
FileCall::FileCall( string d, string i, string ds, string in, string ex ) : dir(d),
id(i), des(ds), inc(in), ext(ex)      {
}
string FileCall::name()       {
       string temp = dir + id + des + inc + ext;
       return temp;   }
void queryDefaults( SIZE& dim, int& num )     {
       InputPar ip;
       cout <<dim;
       cout <<"Realizations:  " <<num <<endl;
       if ( ip.queryStatus() == true )       {
              dim = ip.inputSize();
              num = ip.inputNum();   }}
string queryString()   {
       char inp; string temp;
       cout <<"Input string Y/N" <<endl;
       cin >>inp;
       if ( inp == 'Y' || inp == 'y' )
              cin >>temp;
       else
              temp = "";
       return temp;          }
```

APPENDIX C

DATA FITTING

The data from this work was fit to equations using the software, SigmaPlot 11.0. In Table B.1 these equations are listed. For more information refer to the SigmaPlot help information [102].

Table B.1. Equations for data fitting.

Type	Equation	Parameters
Linear	$y = y_o + ax$	y_o, a
Power	$y = ax^b$	a, b
Gaussian	$y = a\left[\exp\left[-0.5\left(\frac{x - x_o}{b}\right)^2\right]\right]$	a, b, c, x_o
Weibull	$y = y_o + a\left[\exp\left[-0.5\left(\frac{\ln(x/x_o)}{b}\right)^2\right]\right]$	y_o, a, b, x_o

APPENDIX D

MATERIAL PROPERTIES

This appendix lists the Abaqus property files used in the analysis. All the material behaviors are listed, although every property was not necessarily implemented in each analysis. For information about scripting and keywords for Abaqus refer to Abaqus help manuals [98].

YSZ Property Data

```
*Material, name=YSZ
** ----------------------------------------
** UNITS: watts / (mm * C)
*Conductivity
.002,
** ----------------------------------------
** UNITS: g/mm^3
*Density
 0.0590,
** ----------------------------------------
** UNITS: (N/mm^2), -
*Elastic
 216000, 0.315
** ----------------------------------------
** UNITS: % volumetric
*Electrical Conductivity
.252,
** ----------------------------------------
** UNITS: mm/(mm * °C)
*Expansion
8.27e-6, 0
8.5e-6, 100
10.5e-06, 950
*Concrete Damaged Plasticity
56.,  0.,500.,  1.,  0.
*Concrete Compression Hardening
69000.,   0.
    0., 0.01
*Concrete Tension Stiffening
345.00,0.00E+00
247.69,2.50E-03
177.83,5.00E-03
127.67,7.50E-03
91.66, 1.00E-02
65.81, 1.25E-02
47.25, 1.50E-02
33.92, 1.75E-02
24.35, 2.00E-02
17.48, 2.25E-02
12.55, 2.50E-02
9.01,  2.75E-02
6.47,  3.00E-02
4.65,  3.25E-02
*Concrete Tension Damage
0.00,  0.000
0.272, 0.003
0.475, 0.005
0.620, 0.008
0.724, 0.010
0.799, 0.013
```

```
0.853, 0.015
0.892, 0.018
0.919, 0.020
0.939, 0.023
0.954, 0.025
0.964, 0.028
0.971, 0.030
0.977, 0.033
```

Argon/Porosity Property Data

```
*Material, name=PORE
** ---------------------------------------
** UNITS: watts / (mm * C)
*Conductivity
.000016
** ---------------------------------------
** UNITS: g/mm^3
*Density
.001293, 0
.001205, 20
.001127, 40
.001067, 60
.001,  80
.000946, 100
.000898, 120
.000854, 140
.000815, 160
.000779, 180
.000746, 200
.000675, 300
.000566, 350
.000524, 400
** ---------------------------------------
** UNITS: (N/mm^2), -
*Elastic
 1, 0.315
** ---------------------------------------
```

Nickel Property Data

```
*Material, name=NICKEL
** ---------------------------------------
** UNITS: watts / (mm * C)
*Conductivity
    0.0775,   127.
      0.07,   227.
 0.0624393,   307.
 0.0623511,   309.
 0.0622021,   311.
 0.0621209,   313.
 0.0619969,   315.
 0.0619025,   317.
 0.0617648,   319.
 0.0616734,   321.
 0.0615556,   323.
 0.0614642,   325.
 0.0613448,   327.
 0.0612319,   329.
 0.0611418,   331.
 0.0610027,   333.
 0.0608483,   335.
```

Nickel Property Data (cont.)

```
 0.060694,   337.
0.0605368,   339.
0.0603904,   341.
0.0602668,   343.
0.0600925,   345.
0.0599263,   347.
0.0598322,   348.
0.0597391,   349.
0.0596443,   350.
0.0595442,   351.
0.0594663,   352.
0.0594074,  352.5
0.0593553,   353.
0.0593085,  353.5
0.0592492,   354.
0.0592072,  354.5
0.0591474,   355.
0.0590818,  355.5
0.0590826,   356.
0.0591082,  356.5
0.0591426,   357.
 0.059175,  357.5
 0.059184,  357.6
0.0592082,  357.7
0.0592384,  357.8
0.0592889,  357.9
0.0593419,   358.
0.0593853, 358.05
0.0594239,  358.1
0.0594722, 358.15
0.0595289,  358.2
0.0595776, 358.25
0.0596161,  358.3
0.0596566, 358.35
  0.05969,  358.4
 0.059751, 358.45
  0.05983,  358.5
 0.059931, 358.55
 0.059932,  358.6
 0.059932, 358.65
  0.05989,  358.7
 0.059805, 358.75
0.0597009,  358.8
 0.059646, 358.85
0.0595883,  358.9
 0.059548, 358.95
0.0595163,   359.
0.0594632,  359.1
0.0594338,  359.2
 0.059422,  359.3
0.0594072,  359.4
 0.059389,  359.5
0.0593548,   360.
0.0593138,  360.5
0.0592937,   361.
0.0593483,  361.5
  0.05934,   362.
0.0593504,  362.5
0.0593519,   363.
0.0593498,  363.5
0.0593658,   364.
0.0593492,  364.5
```

Nickel Property Data (cont.)

```
 0.0593559,   365.
 0.0593519,  365.5
 0.0593564,   366.
 0.0593706,  366.5
 0.0593776,   367.
  0.059378,  367.5
 0.0593907,   368.
 0.0594734,   369.
 0.0595204,   370.
 0.0595708,   371.
 0.0596163,   373.
 0.0597148,   375.
 0.0597824,   377.
 0.0598622,   379.
 0.0599589,   381.
 0.0600192,   383.
 0.0601024,   385.
 0.0601916,   387.
 0.0602695,   389.
 0.0603566,   391.
 0.0604732,   393.
 0.0605628,   395.
    0.0619,   527.
    0.0638,   627.
    0.0658,   727.
    0.0679,   827.
    0.0697,   927.
    0.0716,  1027.
    0.0735,  1127.
** --------------------------------------
** UNITS: g/mm^3
*Density
 0.0888,
** --------------------------------------
** UNITS: (N/mm^2), -
*Elastic
206617.7014,0.31,25
200297.1506,0.31,100
191700.5364,0.31,200
183001.416,0.31,300
174244.8267,0.31,400
165452.8412,0.31,500
161047.5731,0.31,550
156637.5281,0.31,600
152223.4775,0.31,650
147806.0356,0.31,700
143385.6971,0.31,750
138962.8651,0.31,800
134537.8713,0.31,850
130110.9911,0.31,900
125682.4551,0.31,950
121252.4577,0.31,1000
116821.1642,0.31,1050
112388.7158,0.31,1100
107955.2339,0.31,1150
103520.8235,0.31,1200
** --------------------------------------
** UNITS: mm/(mm * °C)
*Expansion
1.22672E-05,       0
1.24804E-05,      20
1.26951E-05,      40
```

Nickel Property Data (cont.)

```
1.29117E-05,      60
1.31304E-05,      80
1.33514E-05,     100
1.35752E-05,     120
1.38022E-05,     140
1.40331E-05,     160
1.42687E-05,     180
  1.451E-05,     200
1.47586E-05,     220
 1.5017E-05,     240
1.52887E-05,     260
1.55801E-05,     280
1.59039E-05,     300
1.62916E-05,     320
1.68817E-05,     340
 1.7163E-05,     345
1.72447E-05,     346
1.73452E-05,     347
1.74789E-05,     348
1.76903E-05,     349
1.71806E-05,     351
1.69251E-05,     352
1.67713E-05,     353
1.66611E-05,     354
1.65754E-05,     355
1.60128E-05,     375
 1.5895E-05,     395
1.58817E-05,     415
1.59174E-05,     435
1.59821E-05,     455
1.60659E-05,     475
1.61633E-05,     495
 1.6271E-05,     515
1.63867E-05,     535
1.65089E-05,     555
1.66363E-05,     575
1.67681E-05,     595
1.69037E-05,     615
1.70425E-05,     635
1.71842E-05,     655
1.73282E-05,     675
1.74745E-05,     695
1.76228E-05,     715
1.77728E-05,     735
1.79243E-05,     755
1.80773E-05,     775
1.82317E-05,     795
1.83872E-05,     815
1.85438E-05,     835
1.87015E-05,     855
  1.886E-05,     875
1.90195E-05,     895
1.91798E-05,     915
1.93408E-05,     935
1.95026E-05,     955
0.000019665,     975
1.98281E-05,     995
1.99917E-05,    1015
2.01559E-05,    1035
2.03206E-05,    1055
2.04858E-05,    1075
2.06515E-05,    1095
```

Nickel Property Data (cont.)

```
2.08177E-05,    1115
2.09843E-05,    1135
2.11513E-05,    1155
2.13186E-05,    1175
2.14864E-05,    1195
2.16545E-05,    1215
2.18229E-05,    1235
2.19916E-05,    1255
2.21607E-05,    1275
2.23301E-05,    1295
2.24997E-05,    1315
2.26697E-05,    1335
2.28398E-05,    1355
2.30103E-05,    1375
 2.3181E-05,    1395
2.33519E-05,    1415
 2.3523E-05,    1435
** --------------------------------------
** UNITS: kJ/kg*K
*Specific Heat
0.420074, -16.27
 0.425094,  -7.41
 0.428776,   1.81
0.435136,  11.34
  0.44183,  20.99
 0.448943,  29.98
 0.574008,   307.
 0.575541,   309.
 0.576563,   311.
 0.578266,   313.
 0.579629,   315.
 0.581332,   317.
 0.582695,   319.
 0.584568,   321.
 0.586272,   323.
 0.588315,   325.
 0.590189,   327.
 0.592233,   329.
 0.594618,   331.
 0.596662,   333.
 0.598706,   335.
  0.60092,   337.
 0.603304,   339.
  0.60603,   341.
 0.609266,   343.
 0.612332,   345.
 0.615909,   347.
 0.617782,   348.
 0.619826,   349.
 0.622041,   350.
 0.624425,   351.
 0.627321,   352.
 0.628683,  352.5
 0.630216,   353.
  0.63192,  353.5
 0.633623,   354.
 0.635667,  354.5
 0.637711,   355.
 0.639925,  355.5
 0.643161,   356.
 0.647079,  356.5
 0.651678,   357.
```

Nickel Property Data (cont.)

```
0.657299,  357.5
0.658832,  357.6
0.660024,  357.7
0.661727,  357.8
0.663771,  357.9
0.665985,   358.
0.667348, 358.05
0.668711,  358.1
0.670244, 358.15
0.671947,  358.2
 0.67365, 358.25
0.675353,  358.3
0.677227, 358.35
 0.67893,  358.4
0.679612, 358.45
0.678419,  358.5
0.671606, 358.55
0.659172,  358.6
 0.64265, 358.65
 0.63209,  358.7
0.624936, 358.75
0.619997,  358.8
0.616931, 358.85
0.614205,  358.9
 0.61148, 358.95
0.609947,   359.
0.606541,  359.1
0.603815,  359.2
0.602112,  359.3
0.599557,  359.4
0.597684,  359.5
  0.5907,   360.
 0.58542,  360.5
0.581332,   361.
0.578607,  361.5
0.575711,   362.
0.573327,  362.5
0.571112,   363.
0.569068,  363.5
0.567365,   364.
0.565491,  364.5
0.563958,   365.
0.562425,  365.5
0.561063,   366.
0.559871,  366.5
0.558678,   367.
0.557486,  367.5
0.556464,   368.
0.555101,   369.
0.553568,   370.
0.552206,   371.
 0.54931,   373.
0.547266,   375.
0.545222,   377.
0.543519,   379.
0.542156,   381.
0.540623,   383.
0.539431,   385.
0.538409,   387.
0.537387,   389.
0.536536,   391.
0.536025,   393.
```

Nickel Property Data (cont.)

```
 0.535343,   395.
 0.534832,   397.
 0.534492,   399.
 0.533981,   401.
  0.53364,   403.
 0.533299,   405.
 0.533129,   407.
  0.53597,  537.9
  0.54141,  593.5
 0.546849,   649.
  0.55187,  704.6
 0.557309,  760.1
 0.562748,  815.7
 0.568187,  871.3
** ---------------------------------------
*Plastic
179.8,0.00E+00,25
185.56,7.41E-05,25
200.46,3.07E-04,25
220.13,7.19E-04,25
241.53,1.32E-03,25
263.17,2.13E-03,25
284.39,3.13E-03,25
304.95,4.34E-03,25
324.78,5.75E-03,25
343.89,7.36E-03,25
362.33,9.18E-03,25
380.12,1.12E-02,25
397.33,1.34E-02,25
** ---------------------------------------
** UNITS: A ( /s ) , n, t, TEMP (ºC)
*Creep
 3.22e-48,    4.6,     0.,    25.
 7.85e-36,    4.6,     0., 128.57
 1.76e-28,    4.6,     0., 232.14
 1.35e-23,    4.6,     0., 335.71
 4.28e-20,    4.6,     0., 439.29
 1.89e-17,    4.6,     0., 542.86
 2.28e-15,    4.6,     0., 646.43
 1.12e-13,    4.6,     0.,   750.
** ---------------------------------------
```

REFERENCES

1. Teagan, W.P., et al. *Current and future cost structures of fuel cell technology alternatives*. 2000. Oberrohrdorf, Switzerland: European Fuel Cell Forum.

2. Feuer, H. and J. Margalit, *SOFCs-too hot to handle?* American Ceramic Society Bulletin, 2004. **83**(7).

3. Ferguson, J.R., J.M. Fiard, and R. Herbin, *Three-dimensional numerical simulation for various geometries of solid oxide fuel cells.* Journal of Power Sources, 1996. **58**(2): p. 109-122.

4. Stolten, D., D. Froning, and L.G.J. De Haart. *Modelling of planar anode-supported thin-layer SOFC stacks*. 2000. Oberrohrdorf, Switzerland: European Fuel Cell Forum.

5. Yakabe, H. and T. Sakurai, *3D simulation on the current path in planar SOFCs.* Solid State Ionics, Diffusion & Reactions, 2004. **174**(1-4): p. 295-302.

6. Yakabe, H., et al., *3D model calculation for planar SOFC.* Journal of Power Sources, 2001. **102**(1-2): p. 144-154.

7. Roos, M., et al., *Efficient simulation of fuel cell stacks with the volume averaging method.* Journal of Power Sources, 2003. **118**(1-2): p. 86-95.

8. Recknagle, K.P., et al., *Three-dimensional thermo-fluid electrochemical modeling of planar SOFC stacks.* Journal of Power Sources, 2003. **113**(1): p. 109-14.

9. Kakaç, S., A. Pramuanjaroenkij, and X.Y. Zhou, *A review of numerical modeling of solid oxide fuel cells.* International Journal of Hydrogen Energy, 2007. **32**(7): p. 761-786.

10. Khaleel, M.A., et al., *A finite element analysis modeling tool for solid oxide fuel cell development: coupled electrochemistry, thermal and flow analysis in MARC®.* Journal of Power Sources, 2004. **130**(1-2): p. 136-148.

11. Liu, H.-C., et al., *Performance simulation for an anode-supported SOFC using Star-CD code.* Journal of Power Sources, 2007. **167**(2): p. 406-412.

12. Leng, Y.J., et al., *Performance evaluation of anode-supported solid oxide fuel cells with thin film YSZ electrolyte.* International Journal of Hydrogen Energy, 2004. **29**(10): p. 1025-33.

13. Tao, Z., et al., *Stress field and failure probability analysis for the single cell of planar solid oxide fuel cells.* Journal of Power Sources, 2008. **182**(2): p. 540-5.

14. Nakajo, A., et al., *Simulation of thermal stresses in anode-supported solid oxide fuel cell stacks. Part I: Probability of failure of the cells.* Journal of Power Sources, 2009. **193**(1): p. 203-15.

15. Chih-Kuang, L., et al., *Thermal stress analysis of planar solid oxide fuel cell stacks: effects of sealing design.* Journal of Power Sources, 2009. **192**(2): p. 515-24.

16. Chih-Kuang, L., et al., *Thermal stress analysis of a planar SOFC stack.* Journal of Power Sources, 2007. **164**(1): p. 238-51.

17. Xiaohua, D. and A. Petric, *Geometrical modeling of the triple-phase-boundary in solid oxide fuel cells.* Journal of Power Sources, 2005. **140**(2): p. 297-303.

18. Kim, J.H., W.K. Liu, and C. Lee, *Multi-scale solid oxide fuel cell materials modeling.* Computational Mechanics, 2009. **44**(5): p. 683-703.

19. Singhal, S.C. and K. Kendall, *High Temperature Solid Oxide Fuel Cells: Fundamentals, Design, and Applications*. 2003, Oxford, UK: Elsevier Advanced Technology. 405.

20. Dees, D.W., et al., *Conductivity of porous Ni/ZrO2-Y2O3 cermets.* Journal of the Electrochemical Society, 1987. **134**(9): p. 2141.

21. Lee, C.-H., et al., *Microstructure and anodic properties of Ni/YSZ cermets in solid oxide fuel cells.* Solid State Ionics, 1997. **98**(1-2): p. 39.

22. Ji Haeng, Y., et al., *Microstructural effects on the electrical and mechanical properties of Ni-YSZ cermet for SOFC anode.* Journal of Power Sources, 2007. **163**(2): p. 926.

23. San Ping, J. and C. Siew Hwa, *A review of anode materials development in solid oxide fuel cells.* Journal of Materials Science, 2004. **39**(14): p. 4405.

24. Malzbender, J., E. Wessel, and R.W. Steinbrech, *Reduction and re-oxidation of anodes for solid oxide fuel cells.* Solid State Ionics, 2005. **176**(29-30): p. 2201.

25. Radovic, M. and E. Lara-Curzio, *Elastic properties of nickel-based anodes for solid oxide fuel cells as a function of the fraction of reduced NiO.* Journal of the American Ceramic Society, 2004. **87**(12): p. 2242-6.

26. Mori, M., et al., *Thermal expansion of nickel-zirconia anodes in solid oxide fuel cells during fabrication and operation.* Journal of the Electrochemical Society, 1998. **145**(4): p. 1374.

27. Radovic, M., et al., *Thermo-physical properties of Ni-YSZ as a function of temperature and porosity.* Ceramic Engineering and Science Proceedings, 2006. **27**(4): p. 79-85.

28. Kumar, A.N. and B.F. Sorensen, *Fracture resistance and stable crack-growth behavior of 8-mol%-yttria-stabilized zirconia.* Journal of the American Ceramic Society, 2000. **83**(5): p. 1199.

29. Kumar, A.N. and B.F. Sorensen, *Fracture energy and crack growth in surface treated Yttria stabilized Zirconia for SOFC applications.* Materials Science and Engineering A, 2002. **333**(1-2): p. 380.

30. Sorensen, B.F. and A.N. Kumar, *Fracture resistance of 8 mol% yttria stabilized zirconia.* Bulletin of Materials Science, 2001. **24**(2): p. 111.

31. Radovic, M. and E. Lara-Curzio, *Mechanical properties of tape cast nickel-based anode materials for solid oxide fuel cells before and after reduction in hydrogen.* Acta Materialia, 2004. **52**(20): p. 5747.

32. Radovic, M., et al., *Effect of thickness and porosity on the mechanical properties of planar components for solid oxide fuel cells.* Ceramic Engineering and Science Proceedings, 2003. **24**(3): p. 329-336.

33. Radovic, M. and E. Lara-Curzio, *Fracture toughness and slow crack growth behavior of Ni-YSZ and YSZ as a function of porosity and temperature.* Ceramic Engineering and Science Proceedings, 2006. **27**(4): p. 373-381.

34. Lara-Curzio, E., et al., *Effect of thermal cycling and thermal aging on the mechanical properties of, and residual Stresses in, Ni-YSZ/YSZ Bi-layers.* Ceramic Engineering and Science Proceedings, 2006. **27**(4): p. 383-391.

35. Gutierrez-Mora, F., J.M. Ralph, and J.L. Routbort, *High-temperature mechanical properties of anode-supported bilayers.* Solid State Ionics, Diffusion & Reactions, 2002. **149**(3-4): p. 177.

36. Johnson, J. and J. Qu, *Effective modulus and coefficient of thermal expansion of Ni-YSZ porous cermets.* Journal of Power Sources, 2008. **181**(1): p. 85-92.

37. Gokhale, A. and S. Zhang, *ORNL Sample Analysis*, Microstructure images and probability functions. 2008, Georgia Institute of Technology: Atlanta, GA.

38. Pyrz, R., *Quantitative description of the microstructure of composites. Part I: morphology of unidirectional composite systems.* Composites Science and Technology, 1994. **50**(2): p. 197.

39. Lee, K., S. Moorthy, and S. Ghosh, *Multiple scale computational model for damage in composite materials.* Computer Methods in Applied Mechanics and Engineering, 1999. **172**(1-4): p. 175.

40. Li, M., et al., *Three dimensional characterization and modeling of particle reinforced metal matrix composites part II: damage characterization.* Materials Science & Engineering A, 1999. **A266**(1-2): p. 221.

41. Ghosh, S., K. Lee, and P. Raghavan, *A multi-level computational model for multi-scale damage analysis in composite and porous materials.* International Journal of Solids and Structures, 2001. **38**(14): p. 2335.

42. Raghavan, P. and S. Ghosh, *Adaptive multi-scale computational modeling of composite materials.* Computer Modeling in Engineering and Sciences, 2004. **5**(2): p. 151.

43. Raghavan, P., S. Li, and S. Ghosh, *Two scale response and damage modeling of composite materials.* Finite Elements in Analysis and Design, 2004. **40**(12): p. 1619.

44. Swaminathan, S., S. Ghosh, and N.J. Pagano, *Statistically equivalent representative volume elements for unidirectional composite microstructures: Part I - Without damage.* Journal of Composite Materials, 2006. **40**(7): p. 583.

45. Shan, Z. and A.M. Gokhale, *Micromechanics of complex three-dimensional microstructures.* Acta Materialia, 2001. **49**: p. 2001-2015.

46. Shan, Z. and A.M. Gokhale, *Representative volume element for non-uniform micro-structure.* Computational Materials Science, 2002. **24**(3): p. 361.

47. Shan, Z. and A.M. Gokhale, *Digital image analysis and microstructure modeling tools for microstructure sensitive design of materials.* International Journal of Plasticity, 2004. **20**: p. 1347-1370.

48. Tewari, A., et al., *Quantitative characterization of spatial clustering in three-dimensional microstructures using two-point correlation functions.* Acta Materialia, 2004. **52**(2): p. 307.

49. Mao, Y., A.M. Gokhale, and J. Harris, *Computer simulations of realistic microstructures of coarse constituent particles in a hot-rolled aluminum alloy.* Computational Materials Science, 2006. **37**(4): p. 543.

50. Singh, H., et al., *Computer simulations of realistic microstructures of discontinuously reinforced aluminum alloy (DRA) composites.* Acta Materialia, 2006. **54**(8): p. 2131.

51. Singh, H., et al., *Application of digital image processing for implementation of complex realistic particle shapes/morphologies in computer simulated heterogeneous microstructures.* Modelling and Simulation in Materials Science and Engineering, 2006(3): p. 351.

52. Cannillo, V. and W.C. Carter, *A stochastic model of damage accumulation in complex microstructures.* Journal of Materials Science, 2005. **40**(15): p. 3993.

53. Rintoul, M.D. and S. Torquato, *Reconstruction of the structure of dispersions.* Journal of Colloid And Interface Science, 1997. **186**(2): p. 467-476.

54. Yeong, C.L.Y. and S. Torquato, *Reconstructing random media.* Physical Review E, 1998. **57**(1): p. 495.

55. Manwart, C. and R. Hilfer, *Reconstruction of random media using Monte Carlo methods.* Physical Review E, 1999. **59**(5): p. 5596.

56. Cule, D. and S. Torquato, *Generating random media from limited microstructural information via stochastic optimization.* Journal of Applied Physics, 1999. **86**(6): p. 3428.

57. Yeong, C.L.Y. and S. Torquato, *Reconstructing random media. II. Three-dimensional media from two-dimensional cuts.* Physical Review E, 1998. **58**(1): p. 224.

58. Rozman, M.G. and M. Utz, *Efficient reconstruction of multi-phase morphologies from correlation functions.* Physical Review E, 2001. **63**: p. 8.

59. Garmestani, A., *Indepedence of Probability Functions*, J. Johnson, Personal Communication. 2007: Atlanta, GA.

60. GCC, *GCC, the GNU Compiler Collection*, Computer Program. 2008. open source computing software.

61. Debye, P. and A.M. Bueche, *Scattering by an inhomogeneous solid.* Journal of Applied Physics, 1949. **20**: p. 518-525.

62. Debye, P., H.R. Anderson, Jr., and H. Brumberger, *Scattering by an inhomogeneous solid. II. The correlation function and its application.* Journal of Applied Physics, 1957. **28**(6): p. 679-683.

63. Torquato, S., *Random Heterogeneous Materials: Microstructure and Macroscopic Properties*. Interdisciplinary Applied Mathematics. Vol. 16. 2002, New York: Springer. 699.

64. Torquato, S., J.D. Beasley, and Y.C. Chiew, *Two-point cluster function for continuum percolation.* Journal of Chemical Physics, 1988. **88**(10): p. 6540-7.

65. Lee, S.B. and S. Torquato, *Measure of clustering in continuum percolation: computer-simulation of the two-point cluster function.* Journal of Chemical Physics, 1989. **91**(2): p. 1173-8.

66. Sang Bub, L. and S. Torquato, *Pair connectedness and mean cluster size for continuum-percolation models: computer-simulation results.* Journal of Chemical Physics, 1988. **89**(10): p. 6427-33.

67. Lee, S.B., *Connectedness and clustering of two-phase disordered media for adhesive sphere model.* Journal of Chemical Physics, 1993. **98**(10): p. 8119-8119.

68. Yi, Y.B. and A.M. Sastry, *Analytical approximation of the two-dimensional percolation threshold for fields of overlapping ellipses.* Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 2002. **66**(6): p. 1-8.

69. Feng, S., B.I. Halperin, and P.N. Sen, *Transport properties of continuum systems near the percolation threshold.* Physical Review B (Condensed Matter), 1987. **35**(1): p. 197-214.

70. Systat Software, I., *SigmaPlot for Windows*, Computer program. 2008. scientific graphics tool.

71. Garboczi, E.J. and A.R. Day, *An algorithm for computing the effective linear elastic properties of heterogeneous materials: three-dimensional results for composites with equal phase Poisson ratios.* Journal Mechanical Physical Solids, 1995. **43**(9): p. 1349-1362.

72. Drugan, W.J. and J.R. Willis, *Micromechanics-based nonlocal constitutive equation and estimates of representative volume element size for elastic composites.* Journal of the Mechanics and Physics of Solids, 1996. **44**(4): p. 497.

73. Kanit, T., et al., *Apparent and effective physical properties of heterogeneous materials: Representativity of samples of two materials from food industry.* Computer Methods in Applied Mechanics and Engineering, 2006. **195**(33-36): p. 3960.

74. Gitman, I.M., M.B. Gitman, and H. Askes, *Quantification of stochastically stable representative volumes.* Archive of Applied Mechanics, 2006. **75**(2-3): p. 79.

75. Hashin, Z. and S. Shtrikman, *A variational approach to the theory of the elastic behavior of multiphase materials.* Journal Mechanical Physical Solids, 1963. **11**(127-140).

76. Corson, P.B., *Correlation functions for predicting properties of heterogeneous materials. III. Effective elastic moduli of two-phase solids.* Journal of Applied Physics, 1974. **45**(7): p. 3171.

77. Torquato, S. and G. Stell, *Bounds on the effective thermal conductivity of a dispersion of fully penetrable spheres.* International Journal of Engineering Science, 1985. **23**(3): p. 375.

78. Garmestani, H., et al., *Statistical continuum theory for large plastic deformation of polycrystalline materials.* Journal of the Mechanics and Physics of Solids, 2001. **49**(3): p. 589.

79. Segurado, J., C. Gonzalez, and J. Llorca, *A numerical investigation of the effect of particle clustering on the mechanical properties of composites.* Acta Materialia, 2003. **51**(8): p. 2355.

80. Schmauder, S., U. Weber, and E. Soppa, *Computational mechanics of heterogeneous materials - Influence of residual stresses.* Computational Materials Science, 2003. **26**(SUPPL.): p. 142-153.

81. Terada, K., T. Miura, and N. Kikuchi. *Digital image-based modeling applied to the homogenization analysis of intermetallic composites*. in *Computer Aided Assessment and Control of Localized Damage*. 1996. Fukuoka, Jpn: Computational Mechanics Inc, Billerica, MA, USA.

82. Takano, N., et al., *Multi-scale analysis and microscopic stress evaluation for ceramics considering the random microstructures.* JSME International Journal, Series A: Solid Mechanics and Material Engineering, 2003. **46**(4): p. 527.

83. Kumar, H., C.L. Briant, and W.A. Curtin, *Using microstructure reconstruction to model mechanical behavior in complex microstructures.* Mechanics of Materials, 2006. **38**(8-10): p. 818.

84. Mishnaevsky, L., Jr., U. Weber, and S. Schmauder, *Numerical analysis of the effect of microstructures of particle-reinforced metallic materials on the crack growth and fracture resistance.* International Journal of Fracture, 2004. **125**(1-2): p. 33.

85. Mishnaevsky Jr, L.L., *Automatic voxel-based generation of 3D microstructural FE models and its application to the damage analysis of composites.* Materials Science and Engineering A, 2005. **407**(1-2): p. 11.

86. Mishnaevsky, L., Jr., K. Derrien, and D. Baptiste, *Effect of microstructure of particle reinforced composites on the damage evolution: probabilistic and numerical analysis.* Composites Science and Technology, 2004. **64**(12): p. 1805.

87. Selcuk, A. and A. Atkinson, *Strength and toughness of tape-cast yttria-stabilized zirconia.* Journal of the American Ceramic Society, 2000. **83**(8): p. 2029.

88. Faisst, T.A., *Determination of the critical exponent of the linear thermal expansion coefficient of nickel by neutron diffraction.* Journal of Physics: Condensed Matter, 1989. **1**(33): p. 5805.

89. Touloukian, Y.S., et al., *Thermal Expansion: Metallic Elements and Alloys*. Thermophysical Properties of Matter. Vol. 12. 1975, New York: Purdue Research Foundation.

90. Kocks, U.F. and C. Shuh Rong, *On the two distinct effects of thermal activation on plasticity: application to nickel.* Physica Status Solidi A, 1992. **131**(2): p. 403-13.

91. Davis, J.R.J.R. and A.I.H. Committee., eds. *Metals Handbook: Vol 2 Properties and Selection: Nonferrous Alloys and Special-Purpose Materials* Metals Handbook. Vol. 2. 1990, ASM International 1521.

92. Binkele, L., *Significance of discrete Lorenz function levels at high temperatures resulting from new metallic conductivity measurements.* High Temperatures - High Pressures, 1986. **18**(6): p. 599-607.

93. Toolbox, E. *Argon*. 2009 cited; Available from: http://www.engineeringtoolbox.com/thermal-conductivity-d_429.html.

94. Connelly, D.L., J.S. Loomis, and D.E. Mapother, *Specific heat of nickel near the Curie temperature.* Physical Review B (Solid State), 1971. **3**(3): p. 922-32.

95. Ghosh, S., J. Bai, and P. Raghavan, *Concurrent multi-level model for damage evolution in microstructurally debonding composites.* Mechanics of Materials, 2007. **39**(3): p. 241.

96. Gonzalez, C., J. Segurado, and J. Llorca, *Numerical simulation of elasto-plastic deformation of composites: Evolution of stress microfields and implications for homogenization models.* Journal of the Mechanics and Physics of Solids, 2004. **52**(7): p. 1573.

97. Zimmermann, A., W.C. Carter, and E.R. Fuller, Jr., *Damage evolution during microcracking of brittle solids.* Acta Materialia, 2001. **49**(1): p. 127.

98. Pyrz, R., *Correlation of microstructure variability and local stress field in two-phase materials.* Materials Science & Engineering A (Structural Materials: Properties, Microstructure and Processing), 1994. **A177**(1-2): p. 253.

99. Simulia, *Abaqus 6.8.1*, Computer Program. 2009. finite element.

100. Thompson, A.W., *Effect of grain size on work hardening in nickel.* Acta Metallurgica, 1977. **25**(1): p. 83.

101. Thompson, A.W., *Yielding in nickel as a function of grain or cell size.* Acta Metallurgica, 1975. **23**: p. 6.

102. Srinivas, M., G. Malakondaiah, and R. Rao, *Fracture toughness of F.C.C. nickel and strain agening B.C.C. iron in the temperature range 77-773K.* Acta metall. mater, 1993. **41**(4): p. 1301-1312.

103. Yakabe, H., et al., *Evaluation of residual stresses in a SOFC stack.* Journal of Power Sources, 2004. **131**(1-2): p. 278-284.

104. Luton, M.J. and C.M. Sellars, *Dynamic recrystallization in nickel and nickel-iron alloys during high temperature deformation.* Acta Metallurgica, 1969. **17**(8): p. 1033.

105. Weertman, J. and P. Shahinian, *Creep of polycrystalline nickel.* Journal of Metals, 1956. **8**(10, Sec 2): p. 1223.

106. Blum, W. and B. Reppich, *On stress dependence of stationary deformation rate.* 1969. **17**(8): p. 959.

107. Freed, A.D. and K.P. Walker, *Viscoplasticity with creep and plasticity bounds.* International Journal of Plasticity, 1993. **9**(2): p. 213.

108. Norman, E.C. and S.A. Duran, *Steady-state creep of pure polycrystalline nickel from 0.3 to 0.55 Tm.* Acta Metallurgica, 1970. **18**(6): p. 723-731.